uc3m | Universidad Carlos III de Madrid

Grado Universitario en Ingeniería Informática
2017-2018

Trabajo Fin de Grado

“Sistema Big Data Para Procesados Futbolísticos”

Carlos Mayoral Lorenzo

Tutor

Pablo Basanta Val

Leganés, 2018

RESUMEN

En los tiempos actuales cada vez se genera un mayor volumen de datos, y estos datos se almacenan porque son información. El *big data* consiste en procesar todos esos datos para obtener información adicional a la que ya de por sí ofrecen. Gracias a ese procesado de grandes volúmenes de datos se obtienen estadísticas que pueden ser relevantes sobre todo para las grandes compañías, que pueden conocer mejor el uso de sus servicios o el comportamiento de sus usuarios.

Recoger datos para analizarlos posteriormente y obtener información es algo que se hace cada vez más en todos los ámbitos. El problema que surge es cómo analizar una cantidad tan grande de datos. Desde hace tiempo existen herramientas pensadas para este tipo de tareas, pero en pocos años, la cantidad de datos está creciendo exponencialmente, y se necesita mejorar la forma de actuar, se necesitan nuevas herramientas.

En este proyecto se van a utilizar herramientas diseñadas con ese propósito, como *Apache Spark*, y se van a aplicar a eventos deportivos, más concretamente, al seguimiento de un partido de fútbol, un escenario del que se pueden obtener muchos datos y estadísticas que muestren el rendimiento de los equipos y sus jugadores en distintos aspectos del juego.

En este documento se recogen los objetivos del proyecto y la motivación para llevarlo a cabo, además de mostrar el diseño del sistema ideado y explicar su funcionamiento e implementación. También se incluyen distintas pruebas de rendimiento realizadas con sus resultados y las conclusiones obtenidas, además de la planificación del proyecto y su presupuesto.

Palabras clave: Datos, volumen, Big Data, Apache Spark.

ABSTRACT

In current times, a greater volume of data is generated every time, and this data is stored because it is information. The *big data* consists of processing all these data to obtain additional information that they already offer. Thanks to this processing of large volumes of data, statistics are obtained that may be relevant especially for large companies, which may know better the use of their services or the behavior of their users.

Collecting data for later analysis and information obtainment is something that is increasingly done in all areas. The problem that arises is how to analyze such a large amount of data. For a long time, there have been tools designed for this kind of task, but in the last few years, the amount of data has been growing exponentially, so it is necessary to improve the way of getting things done, new tools are needed.

In this project, tools designed for that purpose will be used, such as *Apache Spark*, and will be applied to sports events, more precisely, to the monitoring of a football game, a scenario from which you can obtain lots of data and statistics that show the performance of the teams and their players in different aspects of the game.

This document includes the project objectives and the motivation to carry it out, as well as the design of the devised system and its operation and implementation. It also includes various performance tests performed with their results and conclusions, in addition to the project planning and budget.

Keywords: Data, volume, Big Data, Apache Spark.

CONTENIDO

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1. INTRODUCCIÓN Y OBJETIVOS

1.1 Contexto y motivación

La tecnología está evolucionando y cada vez se aplica a más cosas de la vida. Algo que en su origen consistía puramente en la competitividad, la habilidad, y la capacidad física, como era el deporte, ahora también se está modernizando y desde hace años se están empezando a usar tecnologías ya existentes para aplicarse a los deportes, e incluso se están diseñando nuevas específicamente para ello.

Algunas de estas tecnologías buscan ayudar a los deportistas, para controlar su físico y rendimiento, mientras que otras buscan ayudar a los espectadores que ven los eventos deportivos por televisión o internet, para mostrarles distintas perspectivas del juego y también estadísticas que les ayuden a ver que jugador o equipo está rindiendo mejor.

Cada vez se aplican más tecnologías modernas al mundo de los deportes. Un ejemplo en la actualidad es la implementación del VAR (*Video Assistant Referee*, también conocido como videoarbitraje), un sistema concebido para ayudar a los árbitros durante los partidos que les permite revisar una jugada concreta en un monitor a los pocos segundos de que haya ocurrido [1].

Recoger datos para analizarlos posteriormente y obtener información es algo que se hace cada vez más en todos los ámbitos. Recoger los datos no es difícil, ya que la mayoría de personas utiliza internet y tienen distintas cuentas para distintos servicios. El problema que surge es cómo analizar una cantidad tan grande de datos. Hay que pensar la manera de hacerlo para que lleve el menor tiempo posible y se obtenga la información deseada.

Desde hace tiempo existen herramientas pensadas para este tipo de tareas, pero en pocos años, la cantidad de datos está creciendo exponencialmente, y se necesita mejorar la forma de actuar, se necesitan nuevas herramientas. En este proyecto se van a utilizar herramientas diseñadas con ese propósito, como *Apache Spark*, y se van a aplicar a eventos deportivos, más concretamente, al seguimiento de un partido de fútbol, un escenario del que se pueden obtener muchos datos y estadísticas si se sabe cómo medirlo.

Los espectadores que siguen los partidos por televisión ya disfrutan de características de este tipo, cuando en ciertos momentos durante el partido, y de forma más completa al final de cada parte, se muestran en la pantalla diversos datos sobre el rendimiento de los equipos o de los jugadores en distintos aspectos del juego [2].

Obtener esa información no es tarea fácil. Para ello hay que monitorizar a los jugadores durante el partido, lo que genera un gran flujo de datos que es costoso de analizar en un tiempo reducido. Al tener que procesar una cantidad tan grande de datos, no solo va a ser

necesario configurar la tecnología usada a nuestro gusto, sino que también va a ser necesario adaptar la forma de calcular las estadísticas buscadas a la tecnología utilizada para que el procesado y los cálculos sean eficientes y lleven un tiempo aceptable.

1.2 Objetivos

El objetivo de este proyecto es crear un sistema *big data* eficiente en analizar una gran cantidad de datos derivada de un partido de fútbol para obtener estadísticas referentes al transcurso del juego y al rendimiento de los jugadores, y así demostrar la eficacia de aplicar sistemas basados en eventos para proporcionar un conjunto de analíticas para los entrenadores y managers de los equipos y para los espectadores del evento.

En este proyecto de fin de grado se va a buscar una solución al reto propuesto en el DEBS 2013 Grand Challenge [3], aplicando los fundamentos del *big data* a los datos que aporta para intentar alcanzar la mejor solución posible.

Los objetivos definidos son:

- Estudio del contexto y la situación actual y de las herramientas *big data* disponibles.
- Análisis del problema a resolver y los datos proporcionados.
- Diseño de un sistema *big data* adecuado a los requerimientos del proyecto.
- Implementación del sistema *big data* diseñado.
- Planteamiento y realización de las pruebas de rendimiento con distintas configuraciones.
- Análisis del rendimiento en base a los resultados de las pruebas.
- Obtención de las conclusiones derivadas del análisis de resultados y planteamiento de futuras líneas de trabajo para una posible continuación del proyecto.

1.3 Estructura del documento

- Apartado 1: Introducción y objetivos
  En este apartado se expone la motivación del proyecto realizado y su contexto, además de los objetivos propuestos y la estructura de este documento.
- Apartado 2: Estado del arte
  Este apartado contiene información sobre las tecnologías y las herramientas utilizadas en este proyecto.

- Apartado 3: Diseño del sistema
  En este apartado se muestra el diseño del sistema ideado, sus distintos componentes y las interacciones entre ellos.
- Apartado 4: Implementación
  Este apartado describe los pasos de la implementación del proyecto, el procesado de datos inicial y las posteriores consultas.
- Apartado 5: Uso del sistema
  En este apartado se explican los pasos a seguir para arrancar el sistema y hacerlo funcionar para que se ejecute correctamente.
- Apartado 6: Pruebas y resultados
  Este apartado contiene las pruebas de rendimiento realizadas, sus resultados y un análisis de los mismos.
- Apartado 7: Conclusiones y futuras líneas de trabajo
  En este apartado se exponen las conclusiones obtenidas al final del proyecto y posibles formas de continuar el trabajo realizado.
- Apartado 8: Planificación
  Este apartado cuenta con la planificación establecida para llevar a cabo el proyecto, mostrando la duración de cada fase.
- Apartado 9: Presupuesto e impacto socioeconómico
  En este apartado se calcula el presupuesto que ha sido necesario para la realización del proyecto y se expone la posible influencia que puede tener el proyecto en un futuro próximo.
- Apartado 10: Legislación y marco regulador
  En este apartado se recogen las leyes y los estándares enfocados a proyectos de *big data* como este.
- Apartado 11: Extended abstract
  Este apartado es un resumen del proyecto en inglés.
- Referencias
- Anexos

2. ESTADO DEL ARTE

2.1 Open data

Como ya se ha dicho antes, las nuevas tecnologías quieren cambiar el mundo, pero para ello necesitan probarse hasta tener el funcionamiento deseado y que su lanzamiento de cara a los usuarios no sea un fracaso que dañe su reputación permanentemente y ya no tengan oportunidad de arreglarlo. Este proyecto se basa en un partido donde se han utilizado de antemano unos sensores para que vayan midiendo los movimientos en el partido, y después se han publicado los datos, principalmente pensando en las personas que quieran realizar este proyecto, pero al hacerlo de forma pública también han debido pensar en cualquier otra persona interesada en tener un gran volumen de datos de este tipo, que puedan utilizarse para sus propias tecnologías, o intentar adaptar una tecnología existente a esta clase de datos.

El fin es que sean datos para probar nuevas tecnologías relacionadas con el análisis y/o procesado de grandes volúmenes de datos, y que sirvan a cualquiera que desee utilizarlos para aportar algo que pueda servir en el futuro.

2.2 Big Data

En los tiempos actuales cada vez se genera un mayor volumen de datos. Estos datos se almacenan porque son información. El *big data* consiste en procesar todos esos datos para obtener información adicional a la que ya de por sí ofrecen. Gracias a ese procesado de grandes volúmenes de datos se obtienen estadísticas que pueden ser relevantes sobre todo para las grandes compañías, que pueden conocer mejor el uso de sus servicios o el comportamiento de sus usuarios.

La definición de *big data* según el primer y único estándar que posee hasta la fecha explica que es “un paradigma para permitir la recogida, el almacenamiento, la administración, el análisis y la visualización, potencialmente bajo restricciones de tiempo real, de amplios conjuntos de datos con características heterogéneas” [4].

2.2.1 Las 4V’s

Son cuatro los conceptos principales que definen el *big data*, conocidos como las 4V’s:

- Volumen: El *big data*, como su propio nombre indica, es sinónimo de grandes cantidades de datos. Es su principal característica.

- Velocidad: Analizar una gran cantidad de información nunca ha sido algo que pueda hacerse en poco tiempo, pero con los avances tecnológicos, cada vez se pueden analizar y procesar grandes cantidades de datos con mayor rapidez, incluso en tiempo real, lo cual es una de las razones por las que se está extendiendo su uso.
- Variedad: Los datos que se usan para el *big data* pueden proceder de fuentes muy distintas, estar estructurados de forma distinta o en formatos diferentes. Aun así, una tarea del *big data* es extraer la información relevante de todos los conjuntos de datos, gracias a las herramientas diseñadas para ello.
- Veracidad: Hay que asegurar que los datos que se tienen sean ciertos. De lo contrario, los resultados tampoco serían ciertos en su totalidad y dañaría la utilidad final del proceso entero.

Otro concepto que también se considera a veces es el Valor como se ilustra en la Figura 1, que se define como saber escoger los datos a analizar que puedan aportar más valor a los resultados [5].

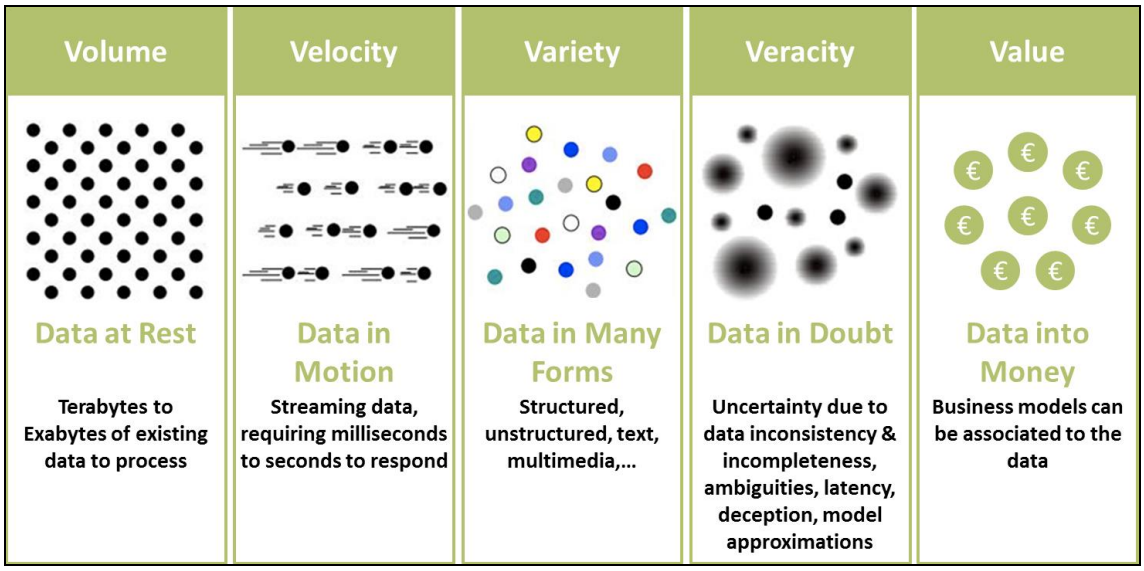

Figura 1: 5V's del big data [6]

2.2.2 Fuentes de datos

En el mundo actual estamos siempre rodeados por tecnología, y cada vez interactuamos más con ella, para distintas cosas. Cada interacción u operación que hagamos es monitorizada y almacenada para poder usarse como información para conocernos mejor. Además, con el actual auge del Internet de las Cosas, cada persona puede ser monitorizada en varios y distintos aspectos [7]. Esto se conoce como dejar un rastro o huella digital [8].

2.2.3 Uso de los datos

La finalidad de los datos recogidos mediante *big data* puede ser muy distinta dependiendo de quién los utilice. En este proyecto, se va a utilizar la información proporcionada para reconstruir un partido de fútbol y analizar distintos aspectos de los jugadores para sacar estadísticas sobre su comportamiento en lo referente al juego.

2.2.4 Análisis de datos

Los diferentes tipos de análisis de los datos se clasifican en cuatro categorías distintas, según la finalidad con la que se hacen [9]:

- Análisis prescriptivo: Se basa en estudiar los datos recopilados para sacar información que pueda servir para detectar rasgos comunes y así mejorar la actuación frente a los problemas en el futuro.
- Análisis predictivo: Consiste en predecir lo que va a pasar estudiando la información recopilada hasta el momento, detectando patrones comunes entre hechos pasados y lo que sucede en el presente.
- Análisis de diagnóstico: Trata de determinar por qué ha sucedido algo, estudiando los datos disponibles sobre ello.
- Análisis descriptivo: Es hacer minería de datos, sacar información útil y relevante de grandes cantidades de datos. Es el tipo de análisis utilizado en este proyecto y se va a explicar en más profundidad en el próximo apartado.

2.2.5 Minería de datos

La minería de datos es el proceso de encontrar patrones en grandes conjuntos de datos mediante el uso de estadística, aprendizaje automático y bases de datos. El objetivo principal del proceso es extraer información de un conjunto de datos para transformarla y estructurarla de manera que pueda ser útil más adelante.

La minería de datos es la parte analítica del proceso KDD (*Knowledge Discovery in Databases*, descubrimiento de información en bases de datos). Este proceso se suele definir con cinco fases: Selección, pre-procesado, transformación, minería de datos y evaluación [10]. No obstante, hay muchas variantes de este esquema, como la CRISP-DM (*Cross Industry Standard Process for Data Mining*), que es la más utilizada y tiene seis fases [11]:

- Comprensión del negocio: Es la fase inicial y se centra en entender los objetivos y los requisitos del proyecto desde una perspectiva empresarial, para transformar esta información en la definición de un problema de minería de datos y un plan para conseguir los objetivos.

- Comprensión de los datos: En esta fase se procesa un conjunto de datos inicial para empezar a interactuar con los datos, identificar problemas de calidad, encontrar las primeras ideas o encontrar subconjuntos que puedan ser interesantes para formar hipótesis para encontrar información oculta.
- Preparación de los datos: Esta fase comprende los procesos para crear el conjunto de datos final, procedente del conjunto inicial de datos, que será utilizado por las herramientas de modelado.
- Modelado: Durante esta fase, se aplican distintas técnicas de modelado, calibradas para tener como parámetros los valores óptimos. Para cada tipo de problema de minería de datos suele haber muchas técnicas diferentes, y algunas necesitan los datos en un formato específico, por lo que a veces es necesario volver a la fase anterior de preparación de los datos.
- Evaluación: Al alcanzar esta fase se debería tener ya un modelo de calidad. Antes de desplegar el modelo, se evalúa y se revisan los pasos llevados a cabo para construirlo, para asegurar que cumple los objetivos empresariales. Uno de los objetivos clave es determinar si hay algún problema empresarial que no se haya considerado con la importancia que merece.
  Al final de esta fase, se debe tomar una decisión acerca del uso de minería de datos en base a los resultados.
- Despliegue: A pesar de que el modelo esté terminado, puede ser necesario más trabajo, como organizar la nueva información que se obtenga con el modelo de manera que sea entendible y útil para el cliente. Dependiendo de los requisitos, la fase de despliegue puede ser más o menos compleja. En la mayoría de casos, el cliente es el que se ocupará del despliegue. En caso de que el despliegue lo haga el analista, el cliente deberá saber qué tiene que hacer para usar el modelo desarrollado.

2.2.6 El futuro del big data

Cada vez hacemos uso de más servicios en internet: redes sociales, productos de entretenimiento, comercio electrónico, etc. Todos ellos requieren que creemos una cuenta particular para poder tener acceso a todo lo que nos ofrecen. Esta es la forma que tienen estos servicios de conocer quienes lo están usando.

Los servicios van aumentado con el tiempo las opciones que nos ofrecen, para intentar que cada uno tenga una experiencia personalizada a su gusto, y en este aspecto, las cuentas en servicios de internet funcionan en ambos sentidos. Por una parte, los usuarios tienen una mejor experiencia de uso, especificando cómo quieren ellos usar cada servicio. Por otra parte, los datos suministrados por los usuarios y las opciones escogidas por cada uno se guardan para ser tratadas. Así cada servicio aprende de nosotros y si lo usamos mucho puede llegar a “conocernos” y saber nuestros gustos, nuestras preferencias o nuestras opiniones. Esta información, aunque al nivel de los usuarios individuales puede ser poco relevante, para las empresas que controlan los servicios es muy valiosa, porque les sirve

para saber las tendencias, los patrones que siguen los consumidores y respecto a las páginas web, también sirve para ofrecer publicidad personalizada a los usuarios, estudiando sus hábitos de consumo.

Desde hace unos años, el número y la capacidad de los servidores repartidos por el mundo ha aumentado enormemente, popularizando el almacenamiento en la nube, que no es otra cosa que la capacidad de almacenar cada vez más datos individuales, lo que a su vez es más información para las empresas, con lo que aumentan su beneficio. Este es el objetivo final de las empresas que usan *big data*: Recolectar datos para adaptarse a los consumidores y beneficiarse con ello.

La gran cadena de comercios Wal-Mart maneja más de un millón de transacciones de usuarios cada hora, acumulando en sus bases de datos aproximadamente 2.5 Petabytes de datos [12]. Wal-Mart ofrece unas tarjetas de socio para sus clientes con las que les ofrece descuentos, y gracias al uso de estas tarjetas, la compañía puede conocer más datos sobre sus clientes y sus compras, lo cual les resulta mucho más valioso que el dinero que dejan de ganar con los descuentos.

El portal de comercio electrónico eBay almacena más de 90 Petabytes de información de las transacciones y el comportamiento de los usuarios. La información se guarda en tres sistemas distintos, donde 40 de los 90 Petabytes se almacenan en un sistema propio para realizar sobre ella análisis en profundidad. La web tiene más de 100 millones de usuarios activos, que generan hasta 100 terabytes de datos cada día, que se almacenan y se utilizan por más de 6000 empleados de eBay [13].

El Gran Colisionador de Hadrones cuenta con aproximadamente 150 millones de sensores que registran 40 millones de datos por segundo. Hay casi 600 millones de colisiones por segundo, cuyos datos, tras filtrarse casi el total, quedan solo 100 colisiones relevantes por segundo. De no filtrarse los datos y almacenar todos sin procesar, habría que almacenar 500 quintillones ($5x10^{20}$) de bytes al día, casi 200 veces más que todas las demás fuentes de datos del mundo juntas [14].

En las carreras de Fórmula 1, cada coche tiene más de cien sensores que transmiten datos en tiempo real y llegan a generar terabytes de información. Se miden varias cosas distintas, como los tiempos de vuelta, la presión de los neumáticos o la eficiencia del combustible. Esta información se transmite a distintos grupos de ingenieros y personal especializado para que la interpreten y hagan los ajustes necesarios para mejorar el rendimiento del coche y su piloto [15].

El servicio de Google, Google Flu Trends, pudo predecir en 2009 la expansión de la pandemia de la gripe A con varios días de antelación, gracias a recopilar y relacionar los datos de distintas búsquedas sobre sus síntomas en distintas fechas y ubicaciones. En Nueva Zelanda compararon los datos de Google Flu Trends con los recogidos por su propio sistema sanitario y confirmaron que eran muy similares [16].

El caso más reciente que demuestra la relevancia del *big data* en el mundo actual es el caso de Cambridge Analytica. Esta compañía es una consultora que contrató el actual presidente de Estados Unidos, Donald Trump, para su campaña electoral para las elecciones de 2016, en las que salió elegido presidente [17].

En 2014, esta empresa obtuvo de manera irregular los datos privados de 87 millones de usuarios de Facebook, la gran mayoría estadounidenses, con el propósito de utilizarlos para sus estrategias políticas y electorales, pensando en las elecciones presidenciales de 2016 [18]. Por supuesto, no se conoce cómo se usaron los datos sustraídos, pero está claro que dieron sus frutos, puesto que Donald Trump ganó las elecciones.

Dejando a un lado el ámbito legal y ético de este caso, sirve para ver la importancia del *big data* en la actualidad; cómo puede suponer una gran ventaja saber utilizarlo y aprovecharse de él. Las empresas ansían tener cada vez más información acerca de los consumidores y los usuarios porque saben que al final significa beneficio para ellos; conocer a los clientes para ofrecerles lo que quieren. En los mejores casos, el beneficio es mutuo y tanto las empresas como los consumidores salen ganando. El *big data* tiene ya una gran importancia en el mundo actual, y viendo los buenos resultados que está dando, es seguro que será cada vez más importante en los próximos años.

2.3 Apache Spark

Apache Spark es un framework de computación en clúster de código abierto. Se desarrolló originalmente en el AMPLab de Berkeley de la Universidad de California, pero su código base fue donado a la Apache Software Foundation, que se encarga desde entonces de mantenerlo. Spark proporciona una interfaz para programar clústeres enteros con paralelismo de datos implícito y tolerancia a fallos.

Apache Spark tiene como fundamento arquitectónico el RDD (*Resilient Distributed Dataset*, conjunto de datos distribuido con capacidad de recuperación), un multi-conjunto de datos de sólo lectura distribuido entre un clúster de máquinas, que se mantiene de manera tolerante a fallos. En las primeras versiones de Spark, la principal API (interfaz de programación de aplicaciones) era RDD, pero a partir de la versión 2.0 se recomienda usar la API Dataset, que ya funciona sobre RDD.

Spark y sus RDD's fueron desarrollados en 2012 como respuesta a las limitaciones en el paradigma de computación en clúster MapReduce, que fuerza una estructura de flujo de datos lineal en programas distribuidos: MapReduce programa datos de entrada desde disco, mapea una función a lo largo de los datos, hace una reducción de los resultados del mapeado, y almacena los resultados de la reducción en disco. Los RDD's de Spark funcionan como una porción de memoria reservada por programas distribuidos, que ofrece una forma restringida de memoria compartida distribuida.

Spark facilita la implementación de algoritmos iterativos y de análisis de datos interactivos o exploratorios. La latencia de estas aplicaciones se puede reducir en varios órdenes de magnitud comparando con una implementación en MapReduce. Entre las clases de algoritmos iterativos están los algoritmos de entrenamiento para sistemas de aprendizaje automático, que fueron uno de los motivos del desarrollo de *Apache Spark*.

Apache Spark requiere un administrador de clústeres y un sistema de almacenamiento distribuido. Para administración de clústeres, Spark cuenta con soporte nativo además de soporte para Hadoop YARN y Apache Mesos. Para almacenamiento distribuido, Spark es compatible con una gran variedad, incluyendo el Sistema de Ficheros Distribuidos de Hadoop (HDFS), Sistema de Ficheros de MapReduce (MapR-FS), Cassandra, OpenStack Swift, Amazon S3, Kudu, y también puede implementarse una solución propia. Spark también soporta un modo local pseudo-distribuido en las que no se necesita almacenamiento distribuido y puede usarse el sistema de ficheros local; un entorno en el que Spark se ejecuta en una sola máquina, con un ejecutor por cada núcleo de CPU [19] [20]. En la Figura 2 se ilustra la variedad de usos que permite Spark.

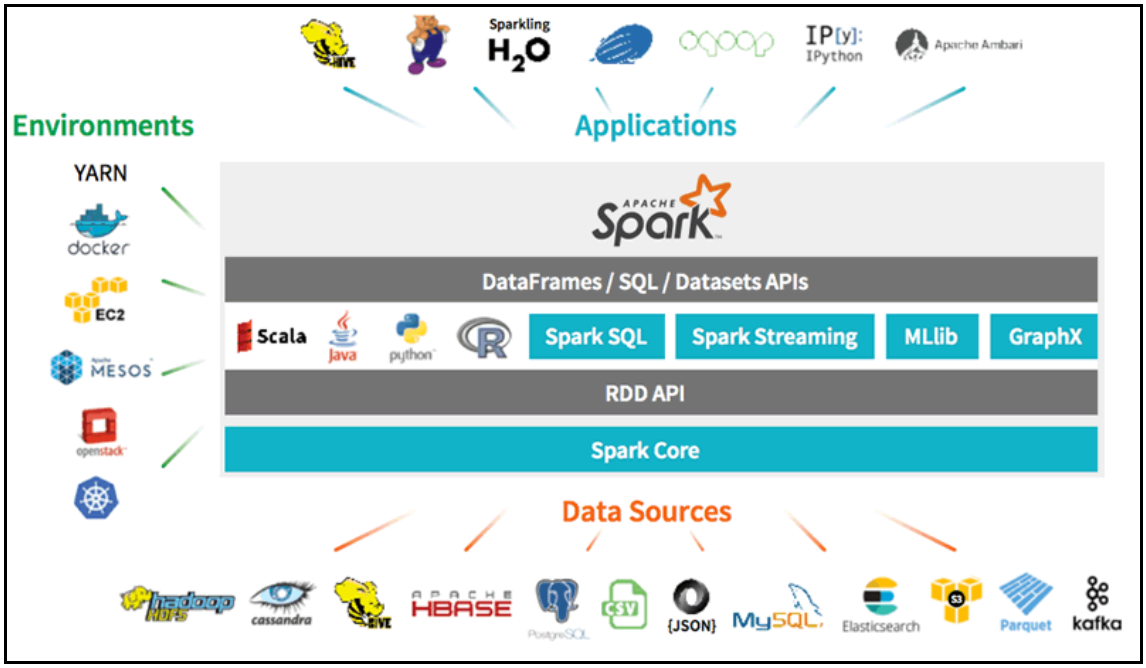

Figura 2: Arquitectura de Spark [21]

2.3.1 Spark Core

Spark Core (el Núcleo de Spark) es la base de todo el proyecto. Proporciona asignación de tareas distribuida, programación y funcionalidades de entrada/salida básicas, accesibles mediante una API (para Java, Python, Scala y R) centrada en la abstracción de RDD. Esta interfaz imita un modelo de programación funcional y de orden superior: un programa controlador solicita operaciones paralelas como mapeado, filtrado o reducción

sobre un RDD pasando una función a Spark, el cual programa la ejecución de la función en paralelo en el clúster. Estas operaciones, y otras adicionales como *joins*, toman RDD's como datos de entrada y generan nuevos RDD's. Los RDD's son inmutables y sus operaciones son de evaluación perezosa: la tolerancia a fallos se logra siguiendo a los sucesores de cada RDD (la secuencia de operaciones que lo han generado) para que puedan reconstruirse en caso de pérdida de datos. Los RDD's pueden contener cualquier tipo de objetos Python, Java o Scala.

Aparte del estilo de programación funcional orientado a RDD, Spark proporciona dos formas restringidas de variables compartidas: las variables *broadcast* (de difusión) para datos de sólo lectura que necesitan estar disponibles para todos los nodos, y acumuladores, para realizar operaciones de reducción de forma imperativa.

2.3.2 Spark SQL

Spark SQL es un componente sobre el Núcleo de Spark que ha introducido una abstracción de datos llamada DataFrames, que da soporte a datos estructurados y semi-esctructurados. Spark SQL proporciona un lenguaje de dominio específico para manipular DataFrames en Scala, Java o Python, y también tiene soporte para lenguaje SQL, con interfaces para línea de comandos y servidores ODBC/JDBC. Aunque los DataFrames carecen de las comprobaciones de tipado en tiempo de compilación de los RDD's, a partir de Spark 2.0, Spark SQL también soporta los fuertemente tipados DataSets [22]. El proceso de ejecución interno de SQL está ilustrado en la Figura 3.

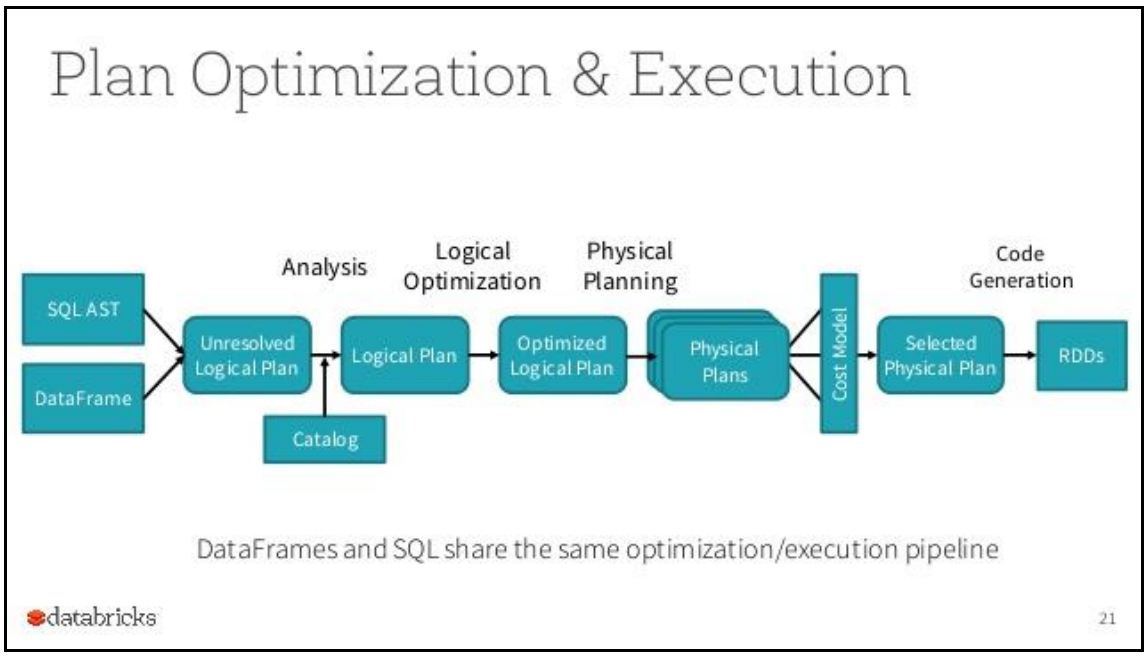

Figura 3: Ejecución de SQL en Spark [23]

En este proyecto, se ha usado Spark en su modo pseudo-distribuido local, utilizándose en un único ordenador y, por tanto, una única CPU, utilizando 4 de los 8 *threads* disponibles al tratarse de una máquina virtual. Permite utilizarse de manera más sencilla, permitiendo centrarse en optimizar el código sin depender de factores externos sin dejar de lado la gran capacidad de Spark para optimizar las consultas y procesar los datos de la manera más rápida posible.

3. DISEÑO DEL SISTEMA

3.1 Introducción

Una vez mostradas las tecnologías que se van a usar en el proyecto, se va a explicar su diseño, donde se describirá su arquitectura, sus núcleos de funcionamiento, sus módulos funcionales y su interacción entre ellos.

3.2 Núcleos de funcionamiento

El sistema diseñado cuenta con un único rol de usuario, que es el de administrador. Éste será el encargado de hacer uso de todas las funcionalidades del sistema y ejecutar las consultas. Se ha establecido sólo un rol porque es un sistema que va a ejecutarse en un único ordenador y es suficiente con un solo usuario para ocuparse de su ejecución, no es necesario contar con más roles.

El administrador del sistema será el encargado de sus funcionalidades, que son: el almacenamiento de los datos, su procesado y la ejecución de las consultas para obtener sus resultados, como queda ilustrado en la Figura 4. A continuación, se explican en más detalle estas funcionalidades que conforman los tres núcleos funcionales del sistema:

- Almacenamiento de datos: El sistema necesita los ficheros con las estadísticas recogidas por los sensores para poder procesarlas y obtener los datos necesarios para las consultas. Estos datos vienen en un fichero de texto plano y se guardarán en el disco duro de la máquina virtual para poder acceder a ellos.
  También se guardarán los datos de las interrupciones del juego recogidas por el árbitro, en formato CSV.
- Procesado de datos y ejecución de las consultas: El fichero almacenado necesita ser leído y procesado para seleccionar los datos que realmente se necesitan para las consultas.
  Una vez seleccionados la consulta y los datos que va a utilizar, se ejecuta el procesado seguido de la consulta.
  Cuando se finaliza la consulta, se escriben los resultados en un fichero de texto en la carpeta designada para ello en el disco duro de la máquina virtual, para que el usuario pueda acceder a ellos.

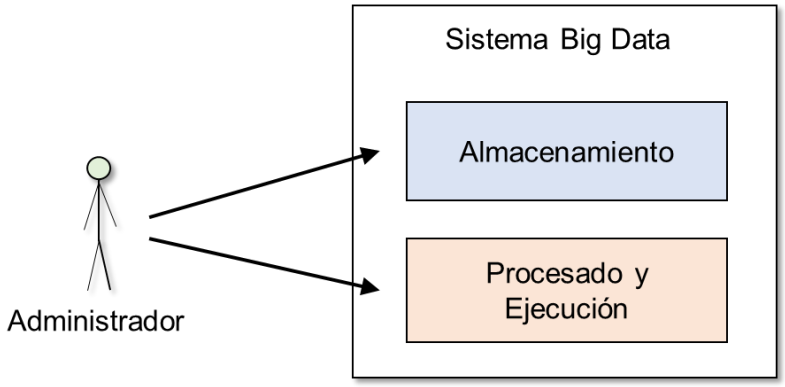

Figura 4: Esquema general del sistema incluyendo los núcleos de funcionamiento

3.3 Arquitectura del sistema

Este sistema se ha pensado para ejecutarse en un único ordenador. Por lo tanto, su diseño no requiere las configuraciones adicionales de un diseño para sistemas distribuidos ni hace uso de funciones de envío y recepción de datos a través de la red. El sistema es único y funciona en la máquina virtual sobre la que se ha instalado, sobre el hardware del ordenador de sobremesa anfitrión.

Los datos que va a necesitar el sistema se van a almacenar en el disco duro de la máquina virtual, en la carpeta "user" que es desde la que se arranca Spark por defecto. Aquí van a estar el fichero "full-game" que contiene las estadísticas de los sensores en formato de texto y pesa poco más de 4 GB y la carpeta "referee-events/Game interruption" que contiene los 2 archivos CSV con las interrupciones del juego, llamados "1st Half" y "2nd half".

Este sistema, al ejecutarse en un único ordenador, tiene un funcionamiento más simple, pero va a sufrir carencias en el rendimiento, ya que un ordenador de sobremesa no está pensado para sistemas *big data* ni para procesado de volúmenes grandes de datos, más aún si se ejecuta el sistema en una máquina virtual, que sólo tiene acceso como mucho a la mitad de las capacidades del ordenador anfitrión.

El disco duro virtual de la máquina virtual usa el disco duro real del ordenador, que en este caso tiene una velocidad de 7200 RPM, que para un archivo de 4 GB como el que se va a usar no va a suponer un problema.

El ordenador cuenta con 16 GB de memoria RAM, pero el sistema sólo tiene acceso a 8 debido a la máquina virtual, lo que ha obligado a ajustar la configuración de Spark a esa cantidad. Esto supone una pérdida de rendimiento al poder tener menos datos disponibles "al vuelo" en la memoria principal y menos espacio para usar como caché en las operaciones intermedias de Spark.

La CPU del ordenador es de gama media-alta, un Intel Core i7-4770. Spark se va a ejecutar en esta única CPU, que cuenta con varios núcleos, por lo que Spark utilizará 1

ejecutor por cada núcleo de CPU. La CPU usada tiene 4 núcleos físicos, cada uno con 2 *threads* de ejecución, que dan lugar a 8 núcleos virtuales. Al usar Spark sobre una máquina virtual, sólo puede acceder a la mitad de recursos del procesador, es decir, 4 núcleos, por lo que Spark utilizará 4 ejecutores para repartir el trabajo entre los núcleos de la CPU, como se ilustra en la Figura 5.

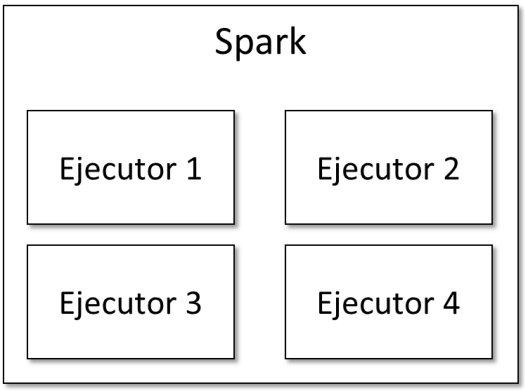

Figura 5: Arquitectura del modo de ejecución de Spark

3.4 Módulos del sistema

El sistema se compone de tres distintas partes o módulos que han de interactuar entre ellos para el completo funcionamiento del sistema:

- Usuario: Es el administrador del sistema y el encargado de interactuar con él. El usuario ha de guardar los datos en el disco duro para que Spark pueda acceder a ellos. El usuario también debe arrancar Spark y especificar los parámetros que utilizará y la consulta que se va a ejecutar.
  Una vez Spark haya completado la consulta especificada, avisará al usuario de su finalización para que pueda acceder a los archivos con los resultados.
- Almacenamiento: Donde se guardan los datos a utilizar y los resultados obtenidos. En el equipo utilizado se refiere al disco duro virtual utilizado por la máquina virtual usada. Este componente es el encargado de almacenar todos los archivos que se van a utilizar para las consultas, para que Spark pueda acceder a ellos, y también los archivos con los resultados, para que el usuario pueda acceder a ellos.
- Spark: Procesa los datos, ejecuta las consultas y guarda sus resultados. Este componente se encarga de leer todos los datos, procesarlos para obtener aquellos necesarios por la consulta que especifique el usuario según los parámetros introducidos, ejecutar dicha consulta siguiendo los parámetros y, una vez finalizada, guardar los resultados en un archivo en el almacenamiento para que sean accesibles para el usuario.

3.5 Interacción entre módulos

A continuación, en la Figura 6 se muestra la relación entre los componentes del sistema para lograr un flujo de actividad correcto.

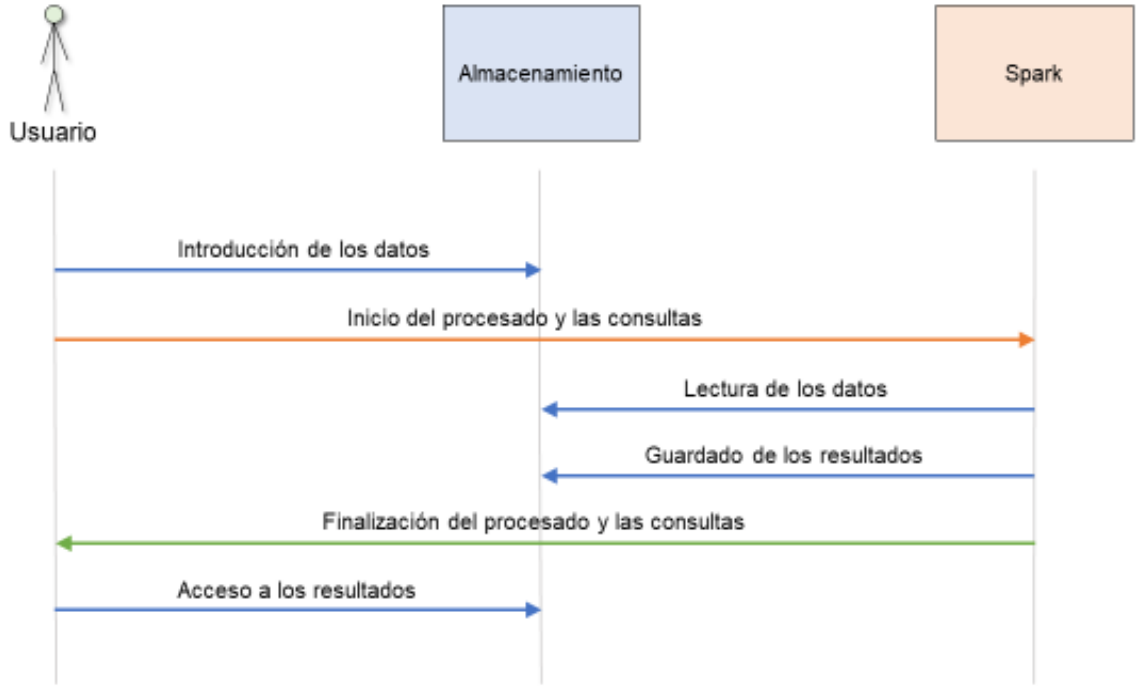

Figura 6: Interacción entre módulos

4. IMPLEMENTACIÓN

4.1 Introducción

En esta parte se va a explicar la preparación del entorno usado para la implementación del proyecto, así como el procesamiento de los datos y la implementación de las consultas.

4.2 Preparación del entorno de implementación

Para tener un mayor control sobre el sistema que va a ejecutar el trabajo y para poder tener también mayor portabilidad, se ha decidido realizar el proyecto sobre una máquina virtual. De esta manera, se pueden realizar cambios en los recursos del sistema para realizar pruebas en distintas condiciones y es fácil recuperar el sistema en caso de fallo grave.

Se ha decidido utilizar el gestor de máquinas virtuales VirtualBox debido a que lleva usándose durante años en los ordenadores de la universidad y los usuarios están ya familiarizados con él. Con él, se ha creado una máquina virtual nueva en la que se ha instalado el sistema operativo Ubuntu 17.04 en su versión de 64 bits. Se ha decidido usar Ubuntu por ser un sistema operativo libre, poco exigente y que otorga gran libertad al usuario. Se ha elegido específicamente la versión 17.04 por ser la versión de Ubuntu con soporte a largo plazo más moderna (en el momento en el que se empezó el proyecto).

También se ha preferido usar Ubuntu en inglés para conservar los nombres originales en las carpetas del sistema.

4.2.1 Instalación de Java

Una vez instalado Ubuntu y configurado con todas las opciones por defecto, el primer paso para hacer funcionar Spark es instalar las últimas versiones de Java, la 7 y la 8. Es muy sencillo de realizar: se abre el terminal, se añade un repositorio si no hay uno por defecto y se introducen los comandos para bajarse la última versión.

```
sudo apt-get install oracle-java7-installer
sudo apt-get install oracle-java8-installer
```

4.2.2 Instalación de Scala

El siguiente paso es instalar el lenguaje de programación Scala, que es el que se usará en Spark. Teniéndolo descargado en formato comprimido, creamos una carpeta nueva para instalarlo y lo descomprimimos ahí. Después, abrimos el bashrc para especificar la ruta donde se ha instalado Scala añadiendo dos nuevas líneas:

```
export SCALA_HOME=/usr/local/src/scala/scala-2.12.4
export PATH=$SCALA_HOME/bin:$PATH
```

4.2.3 Instalación de Git

El último paso antes de instalar Spark es instalar Git, que es tan sencillo como poner el comando apropiado en el terminal.

```
sudo apt-get install git
```

4.2.4 Instalación de Spark

Una vez seguidos los pasos anteriores, deberíamos poder instalar Spark sin problemas. Teniéndolo previamente descargado, especificamos donde queremos descomprimirlo y luego volvemos a abrir el bashrc y añadimos dos líneas para hacer referencia a su carpeta.

```
export SPARK_HOME=/home/carlos/spark
export PATH=$PATH:$SPARK_HOME/bin
```

NOTA: Donde pone “carlos” es el nombre del usuario.

Habiendo hecho correctamente todo, ya se debería poder arrancar Spark con el comando siguiente:

```
spark-shell
```

4.3 Obtención de datos y ficheros sin procesar

Los archivos que contienen los datos a procesar han de obtenerse de la web http://debs.org/debs-2013-grand-challenge-soccer-monitoring. Hay que descargar los archivos comprimidos full-game.gz y referee-events.tar.gz y descomprimirlos posteriormente.

Los datos que se van a procesar están en tres archivos distintos. El primero es un archivo de texto plano llamado “full-game” y es el que tiene todos los datos captados por los

sensores. Es un archivo grande que pesa algo más de 4 GB. Los datos siguen el formato que aparece en la Tabla 1.

CAMPO	EXPLICACIÓN
sid	El identificador numérico de cada sensor
ts	Marca de tiempo (en ms)
x	Posición del sensor en el eje X (en mm)
y	Posición del sensor en el eje Y (en mm)
z	Posición del sensor en el eje Z (en mm)
|v|	Velocidad del sensor (en μm/s)
|a|	Aceleración del sensor (en $\mu m/s^2$)
vx	Vector de dirección de la velocidad en el eje X (de 0 a 10000)
vy	Vector de dirección de la velocidad en el eje Y (de 0 a 10000)
vz	Vector de dirección de la velocidad en el eje Z (de 0 a 10000)
ax	Vector de dirección de la aceleración en el eje X
ay	Vector de dirección de la aceleración en el eje Y
az	Vector de dirección de la aceleración en el eje Z

Tabla 1: Formato de los datos

Los otros dos archivos son CSV llamados “1st Half” y “2nd Half” que se encuentran en la carpeta “referee-events/Game Interruption” y contienen los momentos de pausa y los de reanudación del juego por parte del árbitro, en la primera y la segunda mitad, respectivamente. Los datos siguen el formato que se muestra en la Tabla 2.

CAMPO	EXPLICACIÓN
event_id	Identificador numérico del tipo de evento
event_name	Tipo de evento (comienzo o final de la interrupción)
event_time	Momento de la interrupción/reanudación
event_count	Identificador numérico de cada interrupción y su reanudación
comment	Comentarios adicionales

Tabla 2: Formato de las interrupciones

Estos dos archivos CSV cuentan además con otras estadísticas al final que no son relevantes y que eliminaremos para procesar los datos correctamente.

4.4 Carga de datos

Para poder procesar los ficheros nombrados en el apartado anterior deben copiarse a una carpeta dentro de la máquina virtual.

4.5 Tratado de los datos

Cada una de las cuatro consultas que se han de hacer necesita unos datos específicos de todos los que se ofrecen en los ficheros a procesar. Gran parte de los datos a tratar son comunes entre todas las consultas, pero una vez que se empiezan a transformar los datos para obtener otros derivados de los originales, cada consulta necesita datos distintos, que cada vez van siendo más particulares en cada iteración del procesado de los datos.

4.5.1 Procesamiento general

Hay algunas transformaciones de los datos que siempre han de hacerse al principio para cada una de las cuatro consultas, y son comunes a todas, por lo que siempre se hacen estas transformaciones las primeras y así no hace falta repetirlas en cada consulta.

La primera transformación es para añadir información que se va a necesitar en todas las consultas: el identificador de cada jugador. En la información proporcionada para este proyecto se encuentra la correspondencia entre cada sensor y a qué jugador, balón o árbitro pertenece. Cada balón tiene asignado un único sensor, mientras que el árbitro y la mayoría de jugadores tienen dos, uno para cada pierna, y los porteros tienen otros dos adicionales, uno para cada brazo, ya que pueden utilizarlos en el juego.

Esta transformación es muy sencilla y se hace, una vez teniendo los datos originales en una tabla, añadiendo una columna adicional, que se ha llamado “p” (de ‘player’, jugador en inglés) y que se ha hecho mediante una query de SQL en la que, según el sensor de cada fila, se pone el identificador del jugador asociado.

En la misma query, se puede añadir la segunda transformación que se ha de hacer a los datos. Esta transformación consiste en descartar todas las filas que tengan una marca de tiempo fuera de los tiempos de inicio y final de la primera y de la segunda mitad del partido. Así evitamos contar datos que no son relevantes para las estadísticas y además ahorramos tiempo de ejecución. Para ello, simplemente hay que indicar en la query SQL que los tiempos han de estar entre el ‘ts’ de inicio y el de final de la primera mitad y de

la segunda. Estos tiempos vienen indicados en exactitud en la información proporcionada en la web del proyecto.

Otra transformación usada por las consultas es una derivada de la transformación anterior, y consiste en separar los datos según a la mitad del partido a la que pertenecen. De esta manera, se ahorra tiempo de ejecución ya que en la mayoría de casos sólo se van a realizar cálculos sobre un período de tiempo concreto.

En la mayoría de tablas, se van a ordenar los datos por tiempo, que es una operación que en un primer momento aumenta el tiempo de ejecución, pero después, al operar sobre esas tablas, va a mejorar los tiempos de ejecución en mayor medida que el rendimiento perdido al ordenarlos, ya que las búsquedas son mucho más rápidas. Además, en algunos casos, para mostrar los datos de las consultas de forma correcta hay que ordenarlos necesariamente.

4.5.2 Procesamiento de las interrupciones del juego

Para obtener los momentos en los que se detiene y se reanuda el juego (por parte del árbitro) hay que procesar los ficheros de la carpeta “Game Interruption” y transformarlos para poder combinarlos con los demás datos obtenidos del archivo “full-game” de manera que puedan compararse los tiempos, que es lo más relevante de las interrupciones.

Las interrupciones requieren varias transformaciones. La primera, como ya se ha comentado antes en el apartado 4.3 es eliminar las estadísticas finales de los dos archivos CSV donde vienen las interrupciones. Esto se puede hacer directamente de forma sencilla borrando las últimas líneas donde aparecen dichas estadísticas con un editor de texto.

Al ser dos archivos distintos, pero en el mismo formato, hay que hacer las mismas transformaciones dos veces, una para cada archivo.

Una vez que tenemos los datos de cada archivo en una tabla, eliminamos la primera línea de cada una, ya que es donde vienen los nombres de las columnas, que ya no hacen falta.

También se cambian los nombres de cada campo para hacerlos coincidir con los que ya existen en la tabla de los datos de los sensores. Se han escogido los campos de forma específica para que, si es necesario, puedan combinarse las tablas y se pueda seguir identificando cada interrupción además de distinguir entre marcas de tiempo de sensores y de interrupciones.

Como el tiempo de las interrupciones viene en formato HH:MM:SS:MS hay que transformarlo al mismo formato que se usa en los datos de los sensores, que es en picosegundos. Así que, el primer paso es dividir la columna del tiempo en cuatro columnas distintas: horas, minutos, segundos y milisegundos.

Después se crea otra columna donde se suman los cuatro valores de cada fila pasados a picosegundos, y se borran las cuatro columnas anteriores.

El último paso es sumar a todos los tiempos los picosegundos del momento de inicio de partido. Así tenemos los momentos de cada interrupción en el mismo formato que los tiempos de los sensores, por lo que ya pueden compararse y se pueden combinar los datos cuando se necesite.

Para algunos cálculos de las estadísticas de los jugadores no se van a tener en cuenta estas interrupciones porque, aunque los movimientos y acciones realizados durante estos periodos por los jugadores no afectan al resultado del partido directamente, los jugadores siguen en activo y en movimiento, lo que contribuye en todo momento a su cansancio a medida que avanza el partido y puede ser un factor importante para su rendimiento en los últimos minutos.

4.6 Implementación de las consultas

4.6.1 Primera consulta: Ritmo de carrera de los jugadores

En esta consulta se va a calcular cuánto tiempo pasan los jugadores en ciertos rangos de velocidad, y cuánta distancia recorren en cada velocidad. Las distintas intensidades, de menor a mayor velocidad, son: Parado, al trote, velocidad baja, velocidad media, velocidad alta y esprint.

Calcular esta consulta es algo lento, ya que requiere procesar todos los datos mientras dura el partido, y con el volumen de datos tan grande del que se tiene (cada segundo de juego son 400 líneas de datos por jugador) se ha limitado esta consulta para que se hagan los cálculos solamente sobre un jugador determinado y en un periodo de tiempo definido (cuanto mayor sea el período, mayor será el tiempo de ejecución).

Una vez especificados el jugador y el período de tiempo a analizar, se transforman los momentos de tiempo especificados al formato que usan los sensores (picosegundos transcurridos) para saber qué datos del jugador buscar.

Como cada jugador tiene un sensor en cada pierna, se calcula la media de la velocidad registrada por ambos en cada instante de tiempo, así se tiene una mejor aproximación a la velocidad real del jugador y se simplifican los cálculos posteriores.

Para contar el tiempo y distancia que pasa el jugador en cada intensidad se va a utilizar un bucle, que en Spark tienen la particularidad de no llevarse bien con las variables externas, por lo que, para poder comparar cada valor analizado con el inmediatamente anterior de forma correcta, la solución encontrada es añadir una columna en la tabla de datos en la que va a figurar la velocidad media del valor anterior. Así, en cada línea se tienen los valores suficientes para hacer las comparaciones necesarias.

Para llevar la cuenta de cuánto tiempo pasa el jugador en cada intensidad de carrera y cuánta distancia lleva recorrida en cada una, se utilizan acumuladores de Spark, que en este caso sirven para ir sumando valores en cada iteración del bucle. Se utilizan dos

acumuladores para cada intensidad de carrera, uno para el tiempo transcurrido y otro para la distancia recorrida.

También utilizamos otras variables para guardar en cada iteración los datos y comparar si son duplicados y descartarlos. También guardamos en una variable global el tiempo que se va a añadir por adelantado en cada iteración, ya que gracias a que se conoce la frecuencia de los sensores, sabemos el tiempo que transcurre entre un dato y el siguiente en un mismo jugador. Así no hace falta calcularlo en cada iteración (los márgenes de error son despreciables).

Finalmente, con un bucle *for*, se recorre la tabla con las medias de la velocidad: En cada iteración se calcula la distancia recorrida respecto al dato anterior y se establecen las intensidades actual y anterior del jugador viendo sus velocidades actual y anterior.

Una vez que se conocen las intensidades, se comparan. Si cambia y llevaba más de un segundo en la misma intensidad, se le suma el tiempo transcurrido desde el último cambio a la intensidad anterior. Si cambia, pero llevaba menos de un segundo en esa intensidad, se guarda ese tiempo para sumárselo a la siguiente, hasta que esté más de un segundo seguido en una misma intensidad.

El método utilizado guarda los tiempos solamente cuando cambian las intensidades después de mantenerse iguales más de un segundo, por lo que al final del bucle hay que añadir una comprobación para que, si el momento actual está muy próximo al del final del período especificado para analizar, guarde todos los datos explícitamente.

Al terminar, se escriben los resultados a un archivo de texto.

4.6.2 Segunda consulta: Posesión

Esta consulta va a calcular los tiempos de posesión de cada jugador y el número de veces que golpean el balón (toques) analizando los momentos en los que el balón es golpeado y el tiempo transcurrido entre ellos.

Lo primero es encontrar los momentos en los que el balón aumenta su aceleración, es decir, es golpeado. Para ello, se buscan los instantes en los que la aceleración de la pelota sobrepasa un determinado valor, denominado “precisión”, previamente especificado.

El siguiente paso es buscar al jugador que ha chutado el balón en cada momento de la tabla anterior, es decir, el jugador que está junto al balón (a menos de un metro) en el instante en el que la aceleración crece. Esto se hace de manera fácil, ya que teniendo los instantes de tiempo en los que sucede el golpeo, sólo hay que buscar los datos de los jugadores que sean inmediatamente anteriores y comprobar sus coordenadas con las del balón para saber si están junto a él.

Una vez que tenemos estos datos de los jugadores en una tabla, añadimos a la misma los momentos de interrupción y reanudación del juego. Esta unión se hace con la intención de saber hasta qué momento dura cada posesión en caso de que el juego se detenga si, por ejemplo, el balón sale fuera del campo. Esta limitación a la hora de calcular la posesión

es muy importante debido a las numerosas interrupciones que sufre el partido y es necesario tenerla en cuenta.

Al igual que para la primera consulta, en esta también se va a utilizar un bucle para ir sumando los toques y los tiempos de posesión en cada iteración, por lo que además de usar acumuladores se ha de incluir en cada fila de la tabla la marca de tiempo de la fila anterior. Esto ha de hacerse creando una nueva tabla con cada una de las columnas duplicada pero desplazadas una posición, usando el número de índice de las filas. De esta manera, en cada fila de la tabla tenemos los datos actuales y los inmediatamente siguientes.

Antes del bucle, se crean dos acumuladores por cada jugador: uno para el tiempo de posesión y el otro para el número de toques al balón. También se crean dos acumuladores para guardar el tiempo total de posesión de cada equipo, además de una variable binaria para saber cuándo está detenido el juego (por el árbitro).

Una vez hecho todo lo anterior, se crea un bucle *for* para recorrer la tabla con las columnas duplicadas que lo primero que hace es comprobar que entre cada par de datos próximos haya un mínimo de 0.1 segundos de diferencia. De esta manera, evitamos datos duplicados o falsos positivos, como contar ambas piernas para un solo toque. Se ha escogido ese valor porque se ha comprobado que entre algunos toques diferentes hay tan solo medio segundo de diferencia y se quería escoger el valor más alto intentando perder la mínima cantidad de datos, para intentar encontrar el punto de equilibrio en el que se eliminan la mayoría de falsos positivos y se pierden el menor número de datos relevantes.

Después, el bucle tiene una comprobación por cada jugador en la que suma al jugador y a su equipo correspondiente en cada iteración el tiempo de posesión transcurrido entre el toque al balón actual y el siguiente (que puede ser de otro jugador) y le suma al jugador un toque, siempre que el juego no esté detenido. Al tener en la misma tabla de los golpeos los momentos de detención y reanudación del juego, se ha añadido también una comprobación en cada iteración para que, al detectar un evento de detención del juego, cambie la variable binaria designada para ello y así hace que se omitan los datos de las siguientes iteraciones, hasta que se detecte que se reanuda el juego, y entonces se sigan guardando los datos. Una vez terminado el bucle, todos los datos están guardados en sus acumuladores, y se escriben a un fichero de texto.

4.6.3 Tercera consulta: Mapa de calor

Esta consulta muestra el mapa de calor de un jugador, es decir, las zonas del campo en las que ha estado y cuánto ha pasado en cada una. Es una estadística que cada vez es más usada en los partidos de fútbol profesionales, ya que indica las zonas de influencia de cada jugador.

En esta consulta se ha de especificar el jugador del que se quiere saber el mapa de calor y el periodo (minutos) de tiempo que se quiere analizar. Una vez determinados estos

datos, se pasan los minutos a picosegundos para saber los datos de los sensores que hay que seleccionar y de qué jugador.

El mapa de calor se va a representar mediante celdas, resultado de aplicar una cuadrícula al campo de juego y dividirlo en varias regiones de iguales. Se van a calcular cuatro representaciones distintas del mapa de calor, cada una con un número distinto de celdas en las que se divide el campo, por lo que la configuración que divida el campo en un mayor número de celdas será la más precisa. Estas configuraciones son: 100x64, 50x32, 25x16 y 13x8.

Para cada fila de datos del jugador, sacamos su celda con una simple regla de tres, en la que multiplicamos la X o la Y por el número de celdas total en ese eje y después lo dividimos entre la longitud del campo. Al resultado le aplicamos la función suelo para obtener un valor entero y ese es la celda del eje calculado. Estos cálculos se hacen dos veces por cada fila, una para las X y otra para las Y, en una sola query SQL, y ambas celdas se concatenan en un mismo campo o columna separándolas con un guion.

Ahora, para saber cuánto tiempo ha estado en cada celda en proporción a las demás, se cuentan cuántas filas o datos del jugador seleccionado se tienen en total y cuántas veces ha estado en cada celda con la función *count* de SQL.

Luego, se divide la cantidad de veces que ha estado en cada celda entre el número total de datos y se multiplica por cien para obtener el porcentaje del tiempo que ha pasado el jugador en cada celda.

Al final, ordenamos las celdas de mayor a menor tiempo para que se vea por donde ha pasado más veces y escribimos los resultados en cuatro archivos de texto, uno para cada conjunto de celdas.

4.6.4 Cuarta consulta: Tiros a puerta

Esta consulta va a mostrar los datos del balón en todos los momentos en los que se detecte que puede ser un tiro a puerta. Por tiro a puerta se entiende que el balón se dirige a la portería con cierta velocidad y cruzaría la línea de gol en menos de un segundo y medio, o se marcharía fuera pasando muy cerca de la portería.

Como la trayectoria del balón cambia en cada momento, debido a las fuerzas a las que está sometida la pelota, habría que hacer los cálculos para cada momento de tiempo, lo cual supondría mucho tiempo de ejecución, pero sabiendo que para que haya un tiro a puerta, alguien ha tenido que chutar el balón, se pueden aprovechar los métodos utilizados en la segunda consulta (posesión) para detectar cuando el balón es golpeado con fuerza, y así limitar los cálculos a los momentos posteriores después de esos golpeos, lo que reduce enormemente la cantidad de datos a procesar y por tanto, el tiempo de ejecución.

Para empezar, se buscan los momentos del balón en los que su aceleración crece y sobrepasa el valor de precisión, al igual que en la segunda consulta. (Normalmente,

cuanto menor es el valor, más acertados son los resultados, pero se tardan mucho más en calcular.)

También al igual que en la segunda consulta, en esta necesitamos calcular cuál es el jugador que ha chutado (el más cercano al balón cuando es golpeado) para saber quién ha tirado a puerta.

Una vez que tenemos los momentos del golpeo y sabemos qué jugador chuta, ya podemos hacer los cálculos para identificar si es un tiro a puerta. Para ello, haremos unos simples cálculos de física. Teniendo los datos de la posición del balón, de su velocidad y de la dirección de ésta, podemos calcular si, de seguir a esta velocidad en una trayectoria recta, llegará en menos de un segundo y medio a la línea de gol, y también si por altura entrará en la portería.

Como hemos dicho antes, también consideramos tiros a puerta aquellos que fallan pero pasan cerca de la portería, por lo que los valores para hacer estos cálculos están modificados para añadir un margen a las dimensiones de la portería y contemplar estos casos, además de tener en cuenta también que si el balón está en el aire su trayectoria tenderá a descender.

El método usado para los cálculos puede generar resultados duplicados, por lo que el siguiente paso será filtrar solamente los que tengan valores diferentes con un *select distinct*.

Para terminar, como ya sabemos qué momentos del balón son los que pertenecen a los tiros a puerta, hacemos un *join* de estos datos con el resto de datos del balón en esos momentos para tener una tabla completa con todo, para finalmente escribirlo en un archivo de texto.

5. USO DEL SISTEMA

La máquina virtual usada se ha configurado para utilizar 8 GB de RAM y 4 núcleos del procesador al 100%, y la configuración de Spark utilizada se adapta a estas condiciones. Dicha configuración se ha realizado modificando el archivo spark-defaults.conf en la carpeta "conf" del directorio de instalación de Spark. En este archivo de texto se han añadido las siguientes líneas para intentar aprovechar al máximo la capacidad de la máquina virtual. En algunas ocasiones, incluso ha sido obligatorio aumentar la capacidad indicada inicialmente para poder ejecutar las consultas de mayor volumen sin errores del sistema:

```
spark.driver.maxResultSize          10g
spark.driver.memory                 8g
spark.executor.memory               8g
```

Para poder utilizar los comandos para ejecutar las consultas de la manera más sencilla posible, los archivos que contienen los scripts de ejecución de las consultas deben colocarse en la carpeta Home del usuario de la máquina virtual, que es donde se ha instalado Spark, y en el caso de la máquina utilizada, la carpeta del usuario se llama "carlos".

5.1 Arranque de la máquina

Se ejecuta VirtualBox en Windows, se selecciona la máquina virtual del proyecto y se pulsa en "Iniciar". Aparecerá la pantalla de la máquina virtual mostrando el arranque de Ubuntu hasta que aparezca el escritorio.

5.2 Arranque de Spark

Se abre el terminal de Ubuntu y se introduce "spark-shell" para arrancar Spark. Entonces aparecerán varias líneas de texto del arranque de Spark hasta que aparezca un letrero con "Spark" en grande, lo que significa que ya está arrancado y listo para usar, como se ve en la Figura 7.

Si se quiere comprobar que Spark está funcionando y ver su configuración, se puede abrir un navegador web y acceder a la dirección http://10.0.2.15:4040/environment/

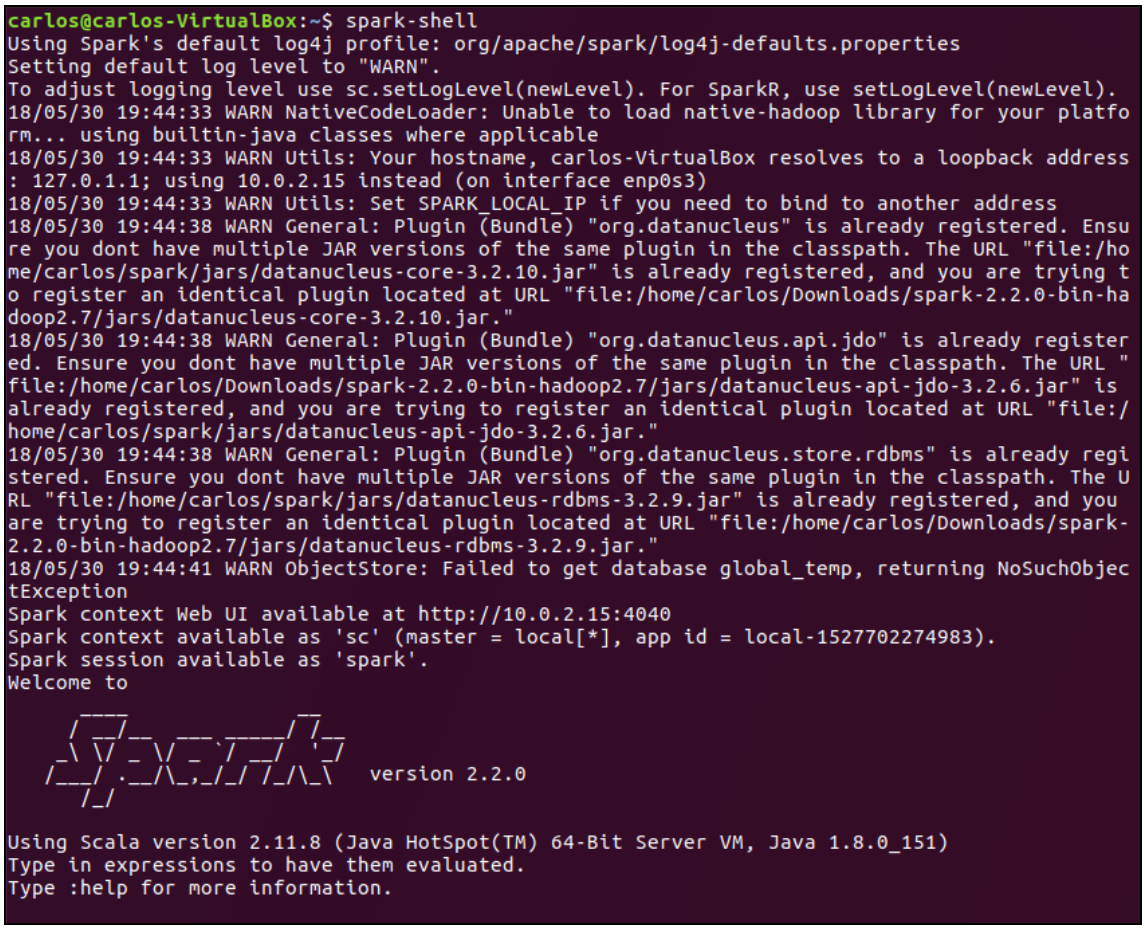

Figura 7: Arranque de Spark

5.3 Ejecución del procesamiento inicial

Para ejecutar el procesamiento y las transformaciones iniciales, necesario para ejecutar las consultas, basta con introducir ":load prep.scala". Esto debería funcionar siempre que el archivo esté ubicado correctamente.

Al terminar de ejecutar este script, aparecerán por pantalla las variables que usará cada consulta y un valor por defecto para cada una, como se muestra en las Figuras 8 y 9. El usuario puede cambiar los valores deseados introduciendo cada variable y un nuevo valor, como por ejemplo "precisión = 150".

```
scala> :load prep.scala
Loading prep.scala...
import org.apache.spark.sql.SparkSession
18/05/30 19:44:52 WARN SparkSession$Builder: Using an existing SparkSession; some configurati
on may not take effect.
res0: org.apache.spark.sql.SparkSession = org.apache.spark.sql.SparkSession@15a591d9
import org.apache.spark.sql._
import org.apache.spark.sql.types._
import org.apache.spark.sql.functions._
import org.apache.spark.sql.SQLContext
import java.io._
defined class Data
defined class Data_p
dataDF: org.apache.spark.sql.DataFrame = [sid: int, ts: bigint ... 11 more fields]
data_p: org.apache.spark.sql.DataFrame = [p: string, sid: int ... 12 more fields]
data_p_h1: org.apache.spark.sql.DataFrame = [p: string, sid: int ... 12 more fields]
data_p_h2: org.apache.spark.sql.DataFrame = [p: string, sid: int ... 12 more fields]
defined class Interruptions
interruptions_1: org.apache.spark.sql.DataFrame = [event_id: string, event_name: string ... 3
 more fields]
interruptions_2: org.apache.spark.sql.DataFrame = [event_id: string, event_name: string ... 3
 more fields]
interruptions_1_data: org.apache.spark.sql.DataFrame = [p: string, event_time_real: string ..
. 1 more field]
interruptions_2_data: org.apache.spark.sql.DataFrame = [p: string, event_time_real: string ..
. 1 more field]
int1_full: org.apache.spark.sql.DataFrame = [p: string, sid: string ... 4 more fields]
int2_full: org.apache.spark.sql.DataFrame = [p: string, sid: string ... 4 more fields]
int1f_added: org.apache.spark.sql.DataFrame = [p: string, sid: string ... 5 more fields]
int2f_added: org.apache.spark.sql.DataFrame = [p: string, sid: string ... 5 more fields]
int1f_ts: org.apache.spark.sql.DataFrame = [p: string, sid: string ... 1 more field]
int2f_ts: org.apache.spark.sql.DataFrame = [p: string, sid: string ... 1 more field]
int1f_ts_real: org.apache.spark.sql.DataFrame = [p: string, sid: string ... 1 more field]
int2f_ts_real: org.apache.spark.sql.DataFrame = [p: string, sid: string ... 1 more field]
```

Figura 8: Procesado inicial (parte 1 de 2)

```
 Preparacion hecha.

 Q1 necesita especificar:
    - player
    - sid
    - half
    - from
    - till

 Q2 necesita especificar:
    - half
    - from
    - till
    - precision
    - test

 Q3 necesita especificar:
    - player
    - half
    - from
    - till

 Q4 necesita especificar:
    - half
    - precision

Valores inicializados por defecto:
player: String = A1
sid: Int = 16
half: Int = 1
from: Int = 0
till: Int = 10
precision: Int = 250
test: Double = 0.1
```

Figura 9: Procesado inicial (parte 2 de 2)

5.4 Ejecución de las consultas

Para ejecutar cada consulta simplemente hay que usar el comando “load” para que Spark cargue y ejecute los archivos con los scripts. Por ejemplo, para ejecutar la consulta 1 hay que introducir “:load q1.scala” y comenzará la ejecución utilizando los parámetros indicados en el apartado anterior, como se ve en las Figuras 10, 12, 13, 15 y 16.

Al terminar la ejecución de una consulta, aparecerá por pantalla el tiempo transcurrido durante la ejecución, como se muestra en las Figuras 11, 14, 15 y 16, y se creará en la carpeta Home una nueva carpeta con el nombre de la consulta donde están los resultados obtenidos, clasificados según los parámetros utilizados para la ejecución.

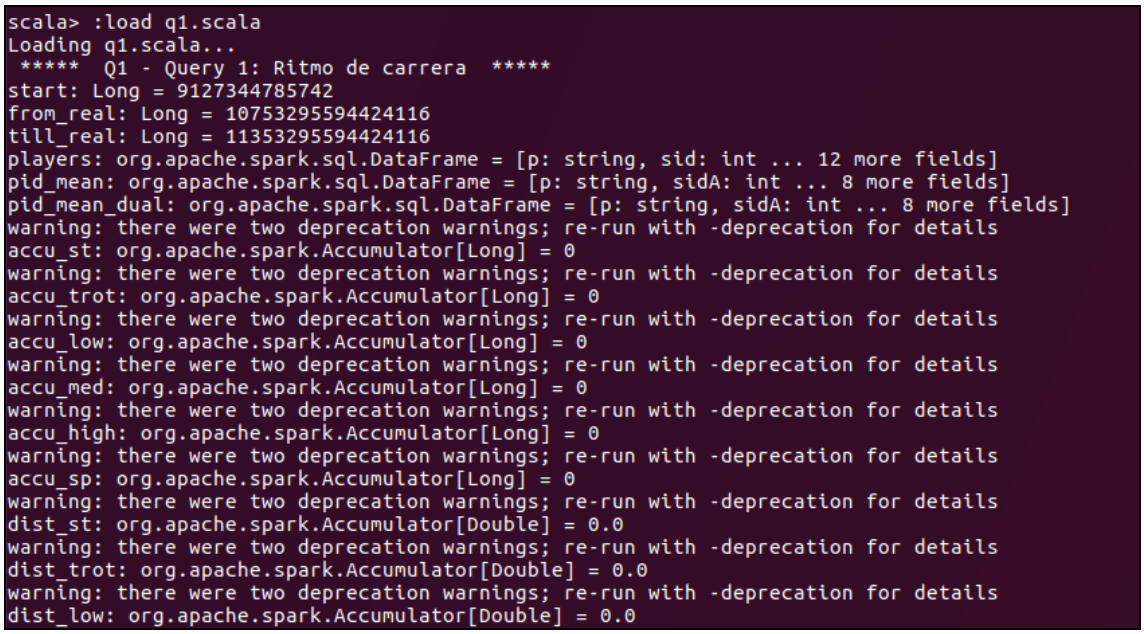

Figura 10: Ejecución de la consulta 1 (parte 1 de 2)

```
warning: there were two deprecation warnings; re-run with -deprecation for details
dist_med: org.apache.spark.Accumulator[Double] = 0.0
warning: there were two deprecation warnings; re-run with -deprecation for details
dist_high: org.apache.spark.Accumulator[Double] = 0.0
warning: there were two deprecation warnings; re-run with -deprecation for details
dist_sp: org.apache.spark.Accumulator[Double] = 0.0
accu_total: Long = 0
dist_total: Double = 0.0
accu_seguido: Long = 0
last_ts1: Long = 0
last_ts2: Long = 0
t: Long = 5000000000
end: Long = 10854143317812
folder: java.io.File = ./q1/A1/half1
res47: Boolean = false
pw: java.io.PrintWriter = java.io.PrintWriter@13e0e957
 *** TIEMPO TRANSCURRIDO: 28.77 minutos ***

Tiempos y distancias de A1 (con sid 16)
desde el minuto 0 hasta el 10 de la 1a mitad:
-STANDING: Time(min) = 2.64, Dist(m) = 280.14416686449454
-TROT: Time(min) = 0.78, Dist(m) = 65.93207186365609
-LOW SPEED: Time(min) = 6.52, Dist(m) = 777.4799007125184
-MEDIUM SPEED: Time(min) = 0.0, Dist(m) = 0.0
-HIGH SPEED: Time(min) = 0.04, Dist(m) = 16.594823209990963
-SPRINT: Time(min) = 0.24, Dist(m) = 51.509920042810165
 TOTAL: 10.244416666666666 mins


scala>
```

Figura 11: Ejecución de la consulta 1 (parte 2 de 2)

```
scala> :load q2.scala
Loading q2.scala...
 *****  Q2 - Query 2: Posesion  *****
start: Long = 11555944715931
separacion: Long = 100000000000
prec_real: Int = 250000000
from_real: Long = 10753295594424116
till_real: Long = 11353295594424116
ball_accelups: org.apache.spark.sql.DataFrame = [sid: int, ts1: bigint ... 5 more fields]
au_nearest: org.apache.spark.sql.DataFrame = [p: string, leg_sid: int ... 4 more fields]
au_nearest_i: org.apache.spark.sql.DataFrame = [p: string, leg_sid: string ... 4 more fields]
auni2: org.apache.spark.sql.DataFrame = [p: string, leg_ts: bigint]
auni3: org.apache.spark.sql.DataFrame = [id: int, p: string ... 1 more field]
auni4: org.apache.spark.sql.DataFrame = [id2: int, player: string ... 1 more field]
auni2dual: org.apache.spark.sql.DataFrame = [p: string, leg_ts1: bigint ... 1 more field]
warning: there were two deprecation warnings; re-run with -deprecation for details
accu_teamA: org.apache.spark.Accumulator[Long] = 0
warning: there were two deprecation warnings; re-run with -deprecation for details
accu_teamB: org.apache.spark.Accumulator[Long] = 0
warning: there were two deprecation warnings; re-run with -deprecation for details
accu_else: org.apache.spark.Accumulator[Long] = 0
warning: there were two deprecation warnings; re-run with -deprecation for details
accu_stopped: org.apache.spark.Accumulator[Long] = 0
warning: there were two deprecation warnings; re-run with -deprecation for details
accu_stopped_real: org.apache.spark.Accumulator[Long] = 0
warning: there were two deprecation warnings; re-run with -deprecation for details
accu_aux: org.apache.spark.Accumulator[Long] = 0
warning: there were two deprecation warnings; re-run with -deprecation for details
accu_teamAe: org.apache.spark.Accumulator[Long] = 0
warning: there were two deprecation warnings; re-run with -deprecation for details
accu_teamBe: org.apache.spark.Accumulator[Long] = 0
warning: there were two deprecation warnings; re-run with -deprecation for details
AGk: org.apache.spark.Accumulator[Long] = 0
warning: there were two deprecation warnings; re-run with -deprecation for details
A1: org.apache.spark.Accumulator[Long] = 0
warning: there were two deprecation warnings; re-run with -deprecation for details
A2: org.apache.spark.Accumulator[Long] = 0
warning: there were two deprecation warnings; re-run with -deprecation for details
A3: org.apache.spark.Accumulator[Long] = 0
warning: there were two deprecation warnings; re-run with -deprecation for details
A4: org.apache.spark.Accumulator[Long] = 0
warning: there were two deprecation warnings; re-run with -deprecation for details
A5: org.apache.spark.Accumulator[Long] = 0
warning: there were two deprecation warnings; re-run with -deprecation for details
A6: org.apache.spark.Accumulator[Long] = 0
```

Figura 12: Ejecución de la consulta 2 (parte 1 de 3)

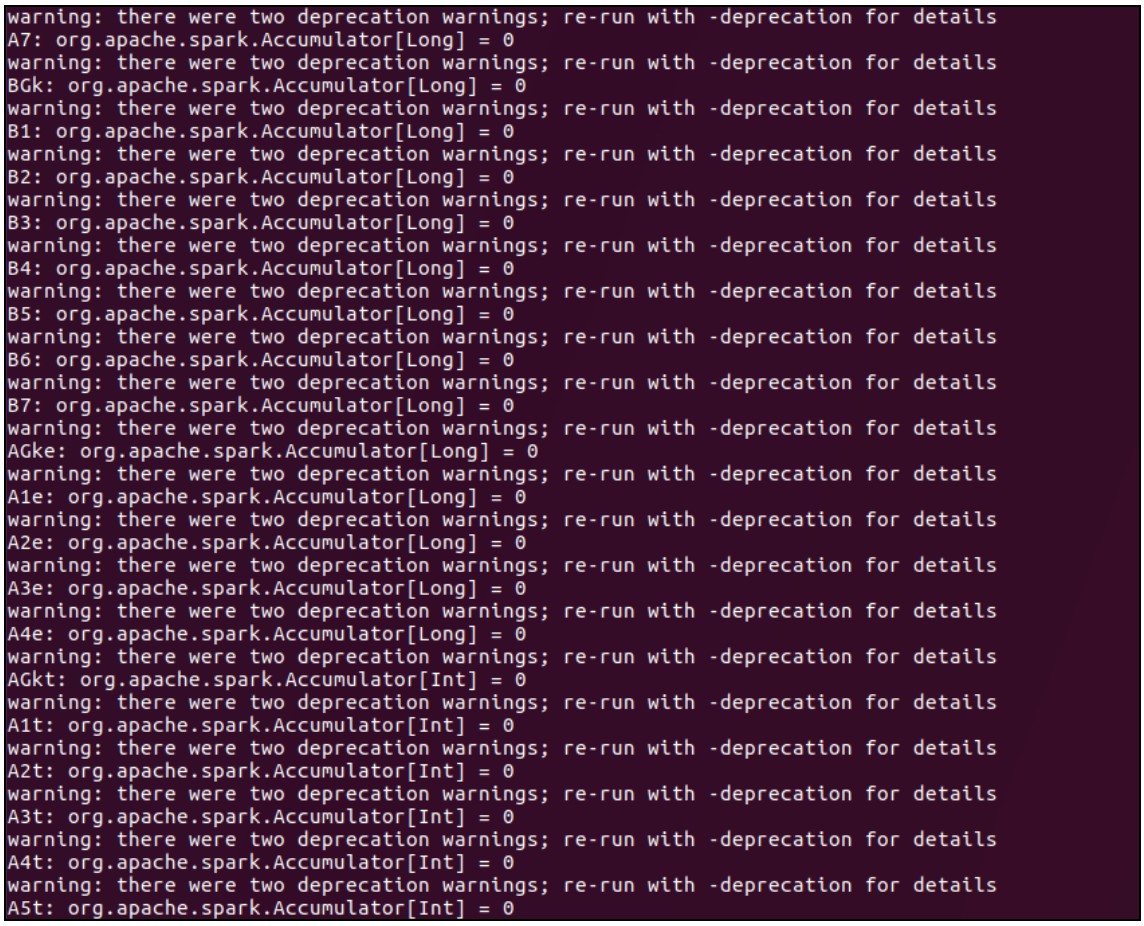

Figura 13: Ejecución de la consulta 2 (parte 2 de 3)

```
warning: there were two deprecation warnings; re-run with -deprecation for details
A6t: org.apache.spark.Accumulator[Int] = 0
warning: there were two deprecation warnings; re-run with -deprecation for details
A7t: org.apache.spark.Accumulator[Int] = 0
warning: there were two deprecation warnings; re-run with -deprecation for details
BGkt: org.apache.spark.Accumulator[Int] = 0
warning: there were two deprecation warnings; re-run with -deprecation for details
B1t: org.apache.spark.Accumulator[Int] = 0
warning: there were two deprecation warnings; re-run with -deprecation for details
B2t: org.apache.spark.Accumulator[Int] = 0
warning: there were two deprecation warnings; re-run with -deprecation for details
B3t: org.apache.spark.Accumulator[Int] = 0
warning: there were two deprecation warnings; re-run with -deprecation for details
B4t: org.apache.spark.Accumulator[Int] = 0
warning: there were two deprecation warnings; re-run with -deprecation for details
B5t: org.apache.spark.Accumulator[Int] = 0
warning: there were two deprecation warnings; re-run with -deprecation for details
B6t: org.apache.spark.Accumulator[Int] = 0
warning: there were two deprecation warnings; re-run with -deprecation for details
B7t: org.apache.spark.Accumulator[Int] = 0
game_stopped: Int = 0
18/05/30 20:26:38 WARN WindowExec: No Partition Defined for Window operation! Moving all data
 to a single partition, this can cause serious performance degradation.
18/05/30 20:26:38 WARN WindowExec: No Partition Defined for Window operation! Moving all data
 to a single partition, this can cause serious performance degradation.
18/05/30 20:26:39 WARN WindowExec: No Partition Defined for Window operation! Moving all data
 to a single partition, this can cause serious performance degradation.
18/05/30 20:26:39 WARN WindowExec: No Partition Defined for Window operation! Moving all data
 to a single partition, this can cause serious performance degradation.
folder: java.io.File = ./q2/half1/0-10
res78: Boolean = false
pw: java.io.PrintWriter = java.io.PrintWriter@1a42e8a3
end: Long = 12749340330590
 *** TIEMPO TRANSCURRIDO: 19.88 minutos ***

scala>
```

Figura 14: Ejecución de la consulta 2 (parte 3 de 3)

```
scala> :load q3.scala
Loading q3.scala...
 *****  Q3 - Query 3: Mapa de calor  *****
start: Long = 12867740568752
from_real: Long = 10753295594424116
till_real: Long = 11353295594424116
player_coord: org.apache.spark.sql.DataFrame = [p: string, sid: int ... 3 more fields]
player_cells_100: org.apache.spark.sql.DataFrame = [p: string, sid: int ... 4 more fields]
player_cells_50: org.apache.spark.sql.DataFrame = [p: string, sid: int ... 4 more fields]
player_cells_25: org.apache.spark.sql.DataFrame = [p: string, sid: int ... 4 more fields]
player_cells_13: org.apache.spark.sql.DataFrame = [p: string, sid: int ... 4 more fields]
player_cells_100_count: org.apache.spark.sql.DataFrame = [cell: string, count: bigint]
player_cells_50_count: org.apache.spark.sql.DataFrame = [cell: string, count: bigint]
player_cells_25_count: org.apache.spark.sql.DataFrame = [cell: string, count: bigint]
player_cells_13_count: org.apache.spark.sql.DataFrame = [cell: string, count: bigint]
sp_count: Long = 237022
player_cells_100_percent: org.apache.spark.sql.DataFrame = [cell: string, count: bigint ... 1
 more field]
player_cells_50_percent: org.apache.spark.sql.DataFrame = [cell: string, count: bigint ... 1
more field]
player_cells_25_percent: org.apache.spark.sql.DataFrame = [cell: string, count: bigint ... 1
more field]
player_cells_13_percent: org.apache.spark.sql.DataFrame = [cell: string, count: bigint ... 1
more field]
player_cells_100_percent_ord: org.apache.spark.sql.DataFrame = [cell: string, count: bigint .
.. 1 more field]
player_cells_50_percent_ord: org.apache.spark.sql.DataFrame = [cell: string, count: bigint ..
. 1 more field]
player_cells_25_percent_ord: org.apache.spark.sql.DataFrame = [cell: string, count: bigint ..
. 1 more field]
player_cells_13_percent_ord: org.apache.spark.sql.DataFrame = [cell: string, count: bigint ..
. 1 more field]
end: Long = 13504129728642
 *** TIEMPO TRANSCURRIDO: 10.6 minutos ***

scala>
```

Figura 15: Ejecución de la consulta 3

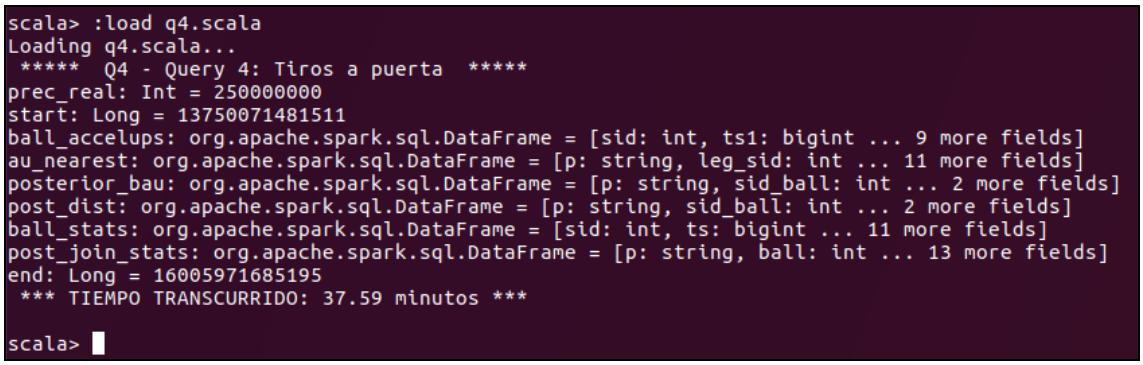

Figura 16: Ejecución de la consulta 4

5.5 Acceso a los resultados

Dependiendo de la consulta, los resultados se guardarán en archivos de distinto formato, pero siempre como texto plano, por lo que pueden abrirse con cualquier editor de texto.

Las rutas donde se guardarán estos archivos dependen de cada consulta y de los parámetros utilizados en su ejecución, por lo que una misma consulta ejecutada dos veces con distintos parámetros, ya sea solo 1 parámetro distinto o más, se guardará en dos rutas diferentes. Esto se ha determinado así para evitar tener que borrar o sobrescribir los archivos de resultados en cada ejecución.

Las rutas para cada consulta, son:

- Consulta 1: Home / q1 / [jugador] / half[mitad] / [periodo].txt
- Consulta 2: Home / q2 / half[mitad] / [periodo] / prec_[precision].txt
- Consulta 3: Home / q3 / [jugador] / [periodo] / cells_[cuadricula] / [archivo de resultados]
- Consulta 4: Home / q4 / half[mitad] / prec_[precision] / [archivo de resultados].csv

donde:

- [jugador] es la identificación de un jugador tal como viene en los datos, por ejemplo, A1.
- [mitad] es la mitad del partido analizada, que puede ser 1 o 2.
- [periodo] es desde qué minuto de una mitad se empieza a analizar y hasta cuál, separados por un guion. Por ejemplo, si se empieza a analizar en el minuto 10 y se termina en el 20, el periodo es 10-20.
- [precisión] es el valor de precisión utilizado, por ejemplo, 150.
- [cuadrícula] es el tamaño de la cuadrícula de celdas en la que se ha dividido el campo para analizarlo, compuesto por el ancho y el alto separados por ‘x’, por ejemplo, 100x64.

- [archivo de resultados] es el nombre asignado por Spark al archivo de texto plano que contiene los resultados. Como el nombre lo asigna Spark, no está determinado cómo se va a llamar y puede tener un nombre de gran longitud con multitud de cifras y letras, difícil de identificar. Aun así, es fácil de localizar, puesto que debe ser el único archivo que aparezca en la ruta establecida, aunque puede aparecer otro archivo creado por Spark para indicar que ha terminado de ejecutar, pero este siempre está vacío.

5.6 Interpretación de los resultados

Se ha intentado sacar los resultados de manera que cumplan con lo que se especifica en el enunciado, a la vez que se encuentren en un formato fácil de interpretar, en la medida de lo posible.

Por la naturaleza de los resultados que han de obtenerse, en algunas consultas no son fáciles de interpretar al verlos, pero tampoco es necesario. Para las consultas 1 y 2, se piden distintas estadísticas sobre los jugadores, que se han podido calcular y representar individualmente en los resultados sin problemas. Pero las consultas 3 y 4, donde se pide obtener cuánto tiempo ha estado un jugador en cada celda y todos los momentos en los que el balón ha sido chutado a portería, respectivamente, tienen unos resultados más complejos de entender.

Los resultados de la consulta 3 son cifras que carecen de sentido si no se interpretan bien, y para hacerlo correctamente, habría que contar con un apoyo más visual e ilustrativo, como una representación gráfica del mapa de celdas. Son datos que en texto plano son difíciles de interpretar. No obstante, sabiendo esta dificultad, en los archivos resultantes, se han ordenado las celdas según el tiempo que ha pasado el jugador en ellas, de mayor a menor, por lo que con solo ver qué celdas son las primeras se podría tener una idea de por qué zonas se ha movido el jugador durante el partido, que es de lo que trata esta consulta.

Los resultados de la consulta 4 son un flujo de datos formado a partir de distintos periodos del partido. Se han recogido los momentos en los que el balón ha formado parte de un posible tiro a puerta, aunque al final no haya acabado yendo a la portería o aunque haya durado muy poco tiempo. A estos grupos de datos del balón se les ha añadido el jugador que chutó el balón, para saber quién es el artífice del tiro a puerta. Tal como están representados los datos en los resultados, no aportan mucha información, aunque sea lo que se pide en la consulta. Para una mejor interpretación de estos resultados, habría que buscar, entre todo el flujo de datos resultante, los momentos en los que hay saltos de tiempo y/o cambia el tirador, para saber cuándo acaba un tiro a puerta y empieza otro. De esta manera se puede ver cuándo y dónde empieza y acaba cada disparo y cuánto dura, que son los datos más relevantes, sin necesidad de revisar todas las demás líneas de datos, que pueden ser muchísimas (recordemos que 1 segundo de juego equivale a 2000 líneas de datos del balón).

Los formatos de los resultados de las consultas 3 y 4 son los siguientes:

- Consulta 3: [Y-X, Cuenta, Porcentaje]
  donde:
  - 'Y' es la posición de la celda en el eje Y (a lo largo del campo).
  - 'X' es la posición de la celda en el eje X (a lo ancho del campo).
  - "Cuenta" es la cuenta de las veces que se ha detectado al jugador en la celda.
  - "Porcentaje" es el porcentaje de tiempo del periodo analizado que ha pasado el jugador en la celda.
- Consulta 4: Jugador, Balón, TS_J, TS_B, x, y, z, |v|, vx, vy, vz, |a|, ax, ay, az
  donde:
  - "Jugador" es el ID del jugador que ha chutado el balón.
  - "Balón" es el SID del balón medido.
  - 'TS_J' es el momento en el que el jugador chuta el balón.
  - 'TS_B' es el momento analizado actualmente.

  El resto de datos son los mismos que los explicados anteriormente en la Tabla 1.

A continuación, en las Figuras 17, 18, 19 y 20, se muestran un ejemplo de los resultados de cada consulta, usando los parámetros establecidos por defecto.

```
Tiempos y distancias de A1 (con sid 16)
desde el minuto 0 hasta el 10 de la 1a mitad:
-STANDING: Time(min) = 2.64, Dist(m) = 280.14416686449454
-TROT: Time(min) = 0.78, Dist(m) = 65.93207186365609
-LOW SPEED: Time(min) = 6.52, Dist(m) = 777.4799007125184
-MEDIUM SPEED: Time(min) = 0.0, Dist(m) = 0.0
-HIGH SPEED: Time(min) = 0.04, Dist(m) = 16.594823209990963
-SPRINT: Time(min) = 0.24, Dist(m) = 51.509920042810165
 TOTAL: 10.244416666666666 mins
```

Figura 17: Resultados de la consulta 1

```
POSESIÓN GENERAL:

-Equipo A = 4.79 min.
-Equipo B = 4.39 min.
(Juego detenido = 1.39 min.)



POSESIÓN INDIVIDUAL

Jugadores equipo A:
-Portero = 24.22 s. / 8 toques
-A1 = 35.39 s. / 18 toques
-A2 = 33.72 s. / 15 toques
-A3 = 97.7 s. / 38 toques
-A4 = 40.11 s. / 23 toques
-A5 = 22.75 s. / 9 toques
-A6 = 20.1 s. / 24 toques
-A7 = 13.71 s. / 8 toques

Jugadores equipo B:
-Portero = 23.61 s. / 8 toques
-B1 = 16.38 s. / 14 toques
-B2 = 38.93 s. / 19 toques
-B3 = 29.64 s. / 18 toques
-B4 = 56.32 s. / 26 toques
-B5 = 49.58 s. / 17 toques
-B6 = 40.57 s. / 8 toques
-B7 = 8.87 s. / 7 toques
```

Figura 18: Resultados de la consulta 2

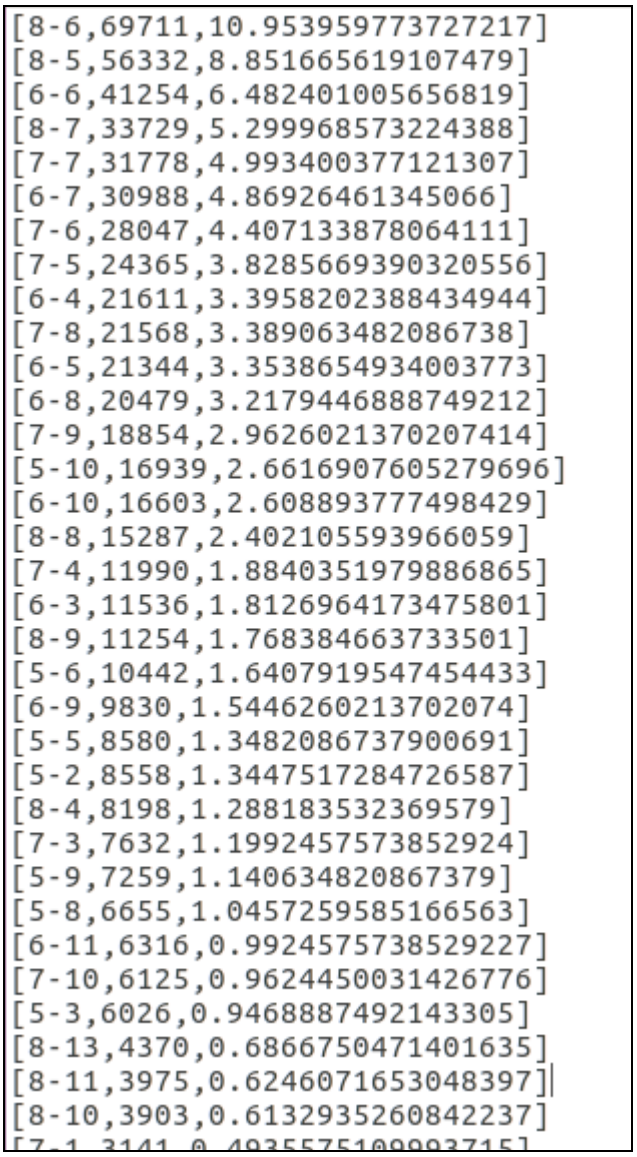

Figura 19: Extracto de los resultados de la consulta 3

```
A3,4,10846802534445482,10846817175788631,11264,-15259,-54,15300146,5720,8153,892,457543575,5929,8037,489
A3,4,10846802534445482,10846817663834146,11268,-15253,-53,15497306,5718,8152,912,455741768,5923,8040,504
A3,4,10846802534445482,10846818151878132,11272,-15249,-55,15503433,5779,8113,878,442154914,6086,7918,497
A3,4,10846802534445482,10846825472550271,11328,-15176,-67,15685921,5957,7992,792,293681396,6702,7395,622
A3,4,10846802534445482,10846826936684206,11339,-15161,-71,15726140,5964,7992,742,272013002,6747,7354,617
B7,4,10940212834826474,10940310443771630,27003,-20703,381,24885998,-1543,-9809,1180,25286145,6933,6319,-3462
B7,4,10940212834826474,10940310931816238,27002,-20715,383,24863589,-1531,-9811,1180,26984279,7195,6137,-3250
B7,4,10940212834826474,10940311419860805,27000,-20727,385,24863453,-1519,-9812,1183,27554819,7599,5704,-3115
B7,4,10940212834826474,10940311907905400,26998,-20739,386,24824961,-1499,-9817,1169,31094517,7822,5480,-2961
B7,4,10940212834826474,10940312395950177,26996,-20751,386,24731310,-1475,-9823,1149,37374481,7645,5838,-2731
B7,4,10940212834826474,10940312883994794,26995,-20763,388,24755597,-1446,-9826,1156,38858444,8405,4804,-2504
B7,4,10940212834826474,10940313372039487,26993,-20775,389,24773386,-1436,-9829,1151,38316994,8650,4311,-2563
B7,4,10940212834826474,10940313860084126,26991,-20786,391,24758418,-1447,-9826,1156,36159218,8372,4791,-2634
B7,4,10940212834826474,10940314348129025,26988,-20797,393,24695436,-1472,-9822,1161,34561465,7298,6271,-2721
B7,4,10940212834826474,10940314836173780,26986,-20809,393,24656073,-1497,-9821,1142,33106471,6116,7275,-3106
B7,4,10940212834826474,10940315324218100,26983,-20820,393,24605366,-1526,-9818,1125,32894376,4418,8309,-3380
B7,4,10940212834826474,10940315812262968,26980,-20832,392,24561205,-1593,-9812,1085,33650470,660,9206,-3847
B7,4,10940212834826474,10940316300307433,26978,-20844,391,24626117,-1614,-9813,1044,29814820,-795,8678,-4903
B7,4,10940212834826474,10940316788352443,26975,-20857,391,24716916,-1636,-9812,1018,25910658,-2835,7493,-5984
B7,4,10940212834826474,10940317276397276,26973,-20869,389,24754809,-1661,-9812,980,26751874,-4509,6248,-6372
B7,4,10940212834826474,10940317764441973,26970,-20881,388,24725846,-1695,-9810,940,31976963,-5515,5910,-5886
B7,4,10940212834826474,10940318252487017,26968,-20893,389,24729911,-1710,-9808,927,33206220,-5977,5539,-5794
B7,4,10940212834826474,10940318740531277,26965,-20904,389,24736960,-1718,-9808,915,33313085,-6174,5210,-5893
B7,4,10940212834826474,10940319228576091,26963,-20916,391,24707753,-1733,-9806,911,35390328,-6306,5413,-5560
B7,4,10940212834826474,10940319716620401,26960,-20927,393,24652068,-1760,-9800,921,39344466,-6456,5880,-4871
B7,4,10940212834826474,10940320204665542,26958,-20939,394,24680082,-1776,-9797,922,39432636,-6993,5299,-4797
B7,4,10940212834826474,10940320692710271,26955,-20951,395,24791959,-1798,-9794,912,39423524,-8105,3316,-4828
B7,4,10940212834826474,10940321180754984,26952,-20963,397,24835274,-1804,-9793,912,38582523,-8359,2529,-4869
B7,4,10940212834826474,10940321668799556,26950,-20975,397,24856492,-1813,-9791,909,38767971,-8504,2108,-4818
```

Figura 20: Extracto de los resultados de la consulta 4

6. PRUEBAS Y RESULTADOS

6.1 Introducción

En esta parte se van a analizar los resultados obtenidos mediante las pruebas de rendimiento, y se va a explicar cómo se han obtenido y los principales factores que les afectan.

6.2 Entornos de prueba

Para comprobar el rendimiento del programa con distintos parámetros de entrada que den distintos resultados en las consultas se han usado combinaciones diferentes de:

- Distintos jugadores.
- Distintas mitades del partido.
- Distintos periodos de tiempo.
- Distintos valores de precisión para detectar golpeos al balón.

Los parámetros usados en las combinaciones se muestran en el siguiente apartado.

6.3 Resultados

En general, los resultados respecto a las distintas combinaciones de parámetros son los esperados. Los parámetros que aumentan el periodo de tiempo a analizar y los que mejoran la precisión tienen un impacto negativo en el rendimiento, como es lógico, ya que aumentan el número de datos a analizar y la consulta tarda más en completarse.

A continuación, se presentan cada una de las cuatro consultas con todos los parámetros utilizados y sus resultados, y cómo los afectan los valores de la precisión.

6.3.1 Consulta 1: Ritmo de carrera

Los distintos parámetros utilizados han sido:

- Mitades: 1, 2.
- Períodos de tiempo (en minutos): [1,15], [1,30].
- Jugadores: A1, A3, B1, B3.

- SID de los jugadores: 16 (A1), 19 (A3), 63 (B1), 67 (B3).

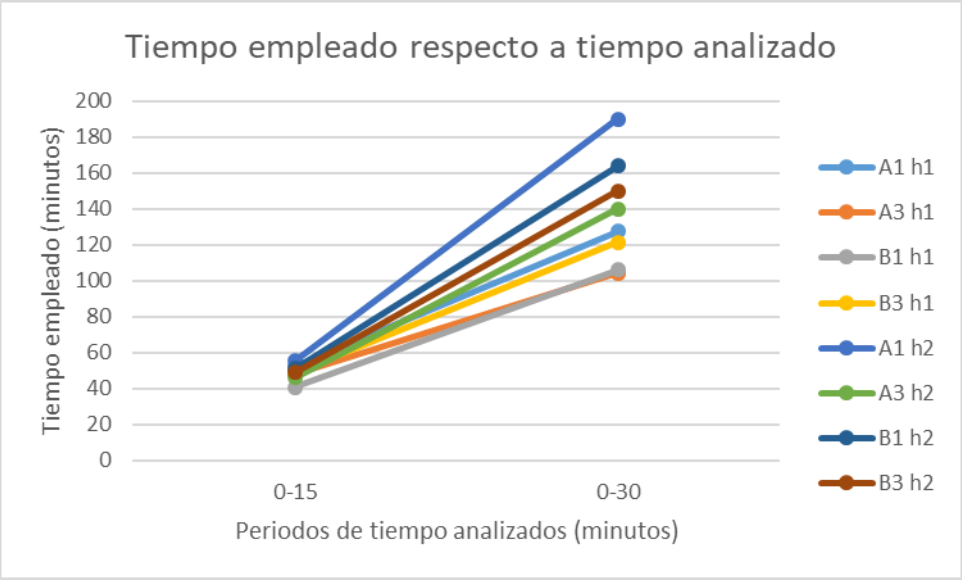

Figura 21: Gráfica de resultados de la consulta 1

Los resultados de las pruebas de rendimiento mostrados en la Figura 21 muestran que se tarda más en analizar cada parte completa del partido que la mitad de cada una, como era lógico esperar. Analizar cada parte completa tarda siempre más del doble de tiempo, e incluso en el peor caso tarda más del triple.

También se ve que procesar la segunda parte del partido siempre tarda más que la primera.

Estas dos observaciones pueden deberse a que, para procesar el archivo original entero, que es lo que se necesita para la segunda parte, hay que recorrer más del doble de datos que para la primera, y además los datos que son necesarios están más lejos en el archivo porque son posteriores. Esto ocurre de manera similar al analizar más datos en una misma parte, aunque sea la primera.

Todo esto va añadiendo tiempo extra de cómputo respecto al análisis de los datos que están primeros en el archivo, sobre todo al analizar cantidades grandes.

Lo último a destacar es que, de los cuatro jugadores analizados (A1, A3, B1, B3), A1 es el que siempre toma más tiempo en ambas partes.

Puesto que las pruebas se han realizado usando el orden de los jugadores: A1, A3, B1, B3; puede que A1 tarde más tiempo que el resto por ser el primero, ya que para analizar los demás, Spark puede usar datos guardados en caché para agilizar el procesado. También puede que, por el ritmo que haya seguido el jugador A1 durante el partido, al ser analizado según los condicionales del código de esta primera consulta, entre en más *if* que el resto, o que entre en los más complejos.

6.3.2 Consulta 2: Posesión

Los distintos parámetros utilizados han sido:

- Mitades: 1, 2.
- Períodos de tiempo (en minutos): [1,10], [1,20], [1,30].
- Valores de la precisión (en m/s^2): 300, 250, 200, 150, 100, 75, 55.

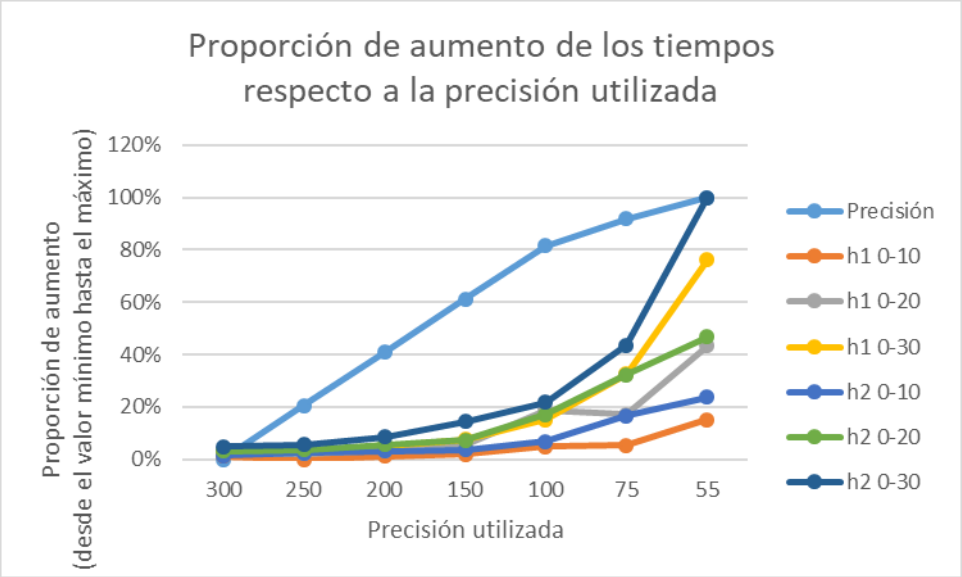

Figura 22: Gráfica de resultados de la consulta 2

En la gráfica de los resultados de la Figura 22 se ve claramente como el rendimiento de la consulta va escalado a la cantidad de datos analizados y a la precisión usada, y también depende de la mitad del partido analizada.

Cuantos más minutos hay que analizar, son más datos que procesar, y, por lo tanto, se tarda más tiempo. También se ve que lo que más influye en el tiempo que tarda es la precisión. Con valores de precisión altos (baja precisión) los tiempos apenas se ven afectados, pero conforme se va reduciendo ese valor, el tiempo aumenta en gran medida, incluso parece aumentar exponencialmente en los casos con más datos a analizar. El efecto de la precisión sobre los resultados se analizará en mayor profundidad más adelante, en el siguiente apartado.

También se ve que los análisis de la primera parte siempre tardan menos que los análisis equivalentes de la segunda mitad. Lo más probable es que esto se deba a las mismas causas que en la primera consulta, donde ocurría lo mismo. Para analizar la segunda mitad hay que recorrer más del doble de los datos que para la primera y estos datos están más lejos en el archivo.

En el enunciado del proyecto se menciona que al final de la primera parte, el sensor del balón dejó de funcionar, y no se pudieron medir sus datos durante los últimos dos minutos y medio. Esta puede ser una de las causas por las que el total de los tiempos analizados no coincida exactamente con los minutos jugados, y otra causa más de que los análisis de la segunda mitad del partido tarden más que los de la primera, ya que ésta tiene menos datos que analizar.

6.3.3 Consulta 3: Mapa de calor

Los distintos parámetros utilizados han sido:

- Mitades: 1, 2.
- Jugadores: A1, A2, A3, B1, B2, B3.
- Períodos de tiempo para el equipo A (en minutos): [1,15], [1,30].
- Períodos de tiempo para el equipo B (en minutos): [1,10], [1,20], [1,30].

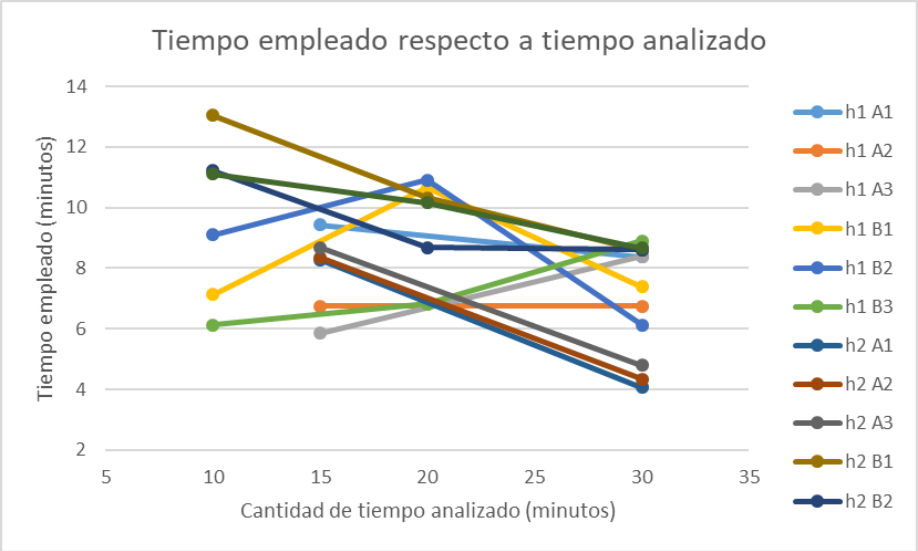

Figura 23: Gráfica de resultados de la consulta 3

Viendo la gráfica de la Figura 23, los resultados son muy variados. La única tendencia apreciable es que el tiempo se reduce al analizar más cantidad de datos. Los tiempos máximos y mínimos analizando 10 minutos son 13 y 6, respectivamente, y analizando 30 minutos son 9 y 4.

La única explicación posible es que tarde menos por tener datos anteriores en la caché de Spark y así el programa no necesite volver a buscarlos.

6.3.4 Consulta 4: Tiros a puerta

Los distintos parámetros utilizados han sido:

- Mitades: 1, 2.
- Valores de la precisión (en m/s^2): 300, 250, 200, 150, 100, 75, 55.

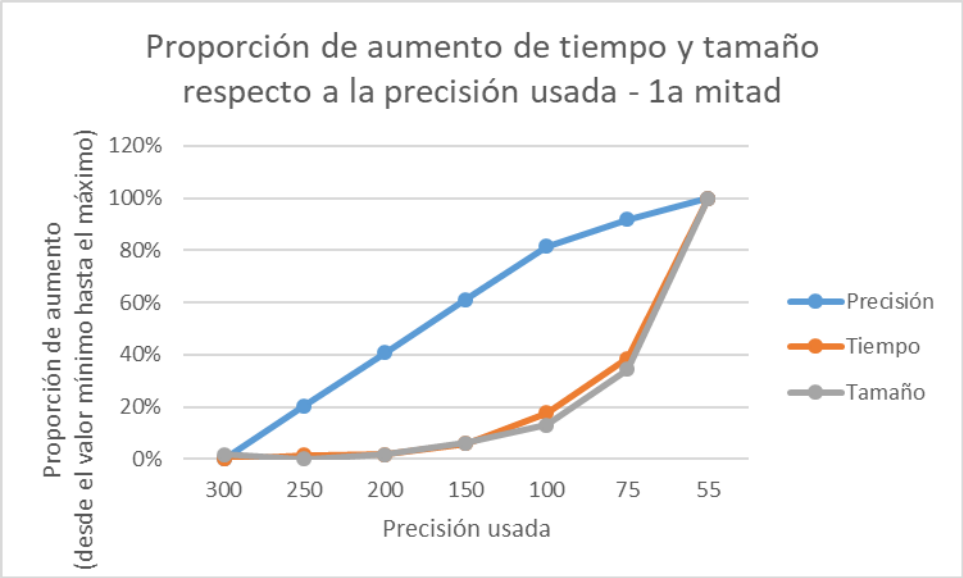

Figura 24: Gráfica 1 de resultados de la consulta 4

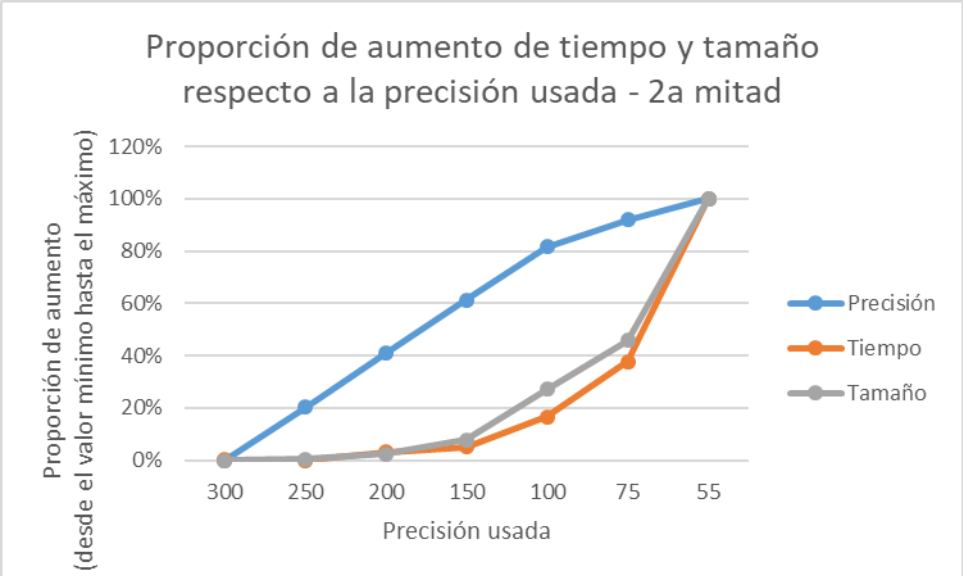

Figura 25: Gráfica 2 de resultados de la consulta 4

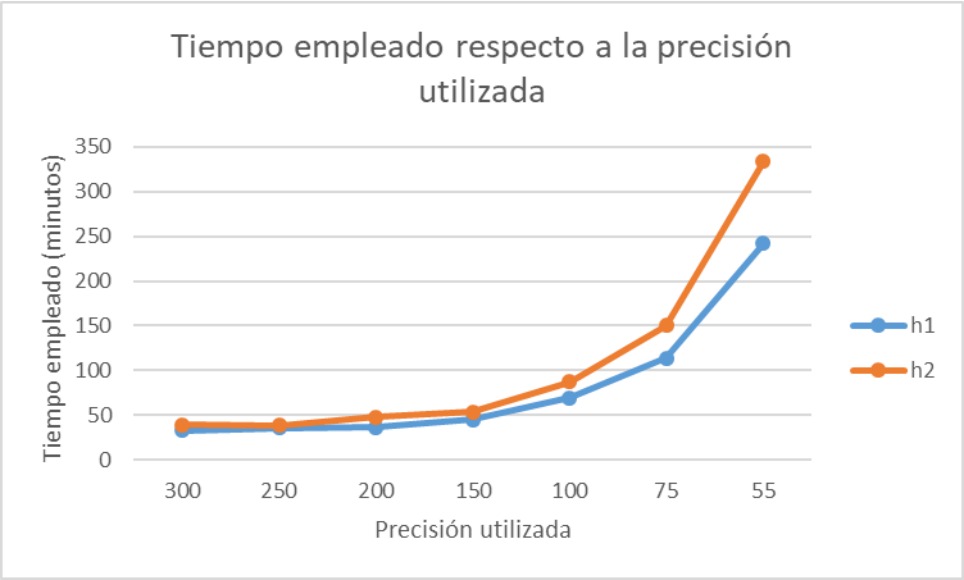

Figura 26: Gráfica 3 de resultados de la consulta 4

En las gráficas de las Figuras 24, 25 y 26 obtenidas de los resultados de las pruebas de rendimiento se puede ver fácilmente que el tamaño del fichero generado es casi idénticamente proporcional al tiempo que tarda en procesarse la consulta.

Esto tiene sentido, ya que el tiempo que se tarda en procesar la consulta depende principalmente de la cantidad de datos procesados, que son los que aparecen en el fichero resultante, por lo que es lógico pensar que el tiempo aumentará de forma casi equivalente al aumento de los datos y el tamaño del fichero.

También se observa cómo, tanto el tiempo empleado como el tamaño del archivo, aumentan con la precisión utilizada. Con los valores de precisión más bajos (mayor precisión) aumentan en una proporción mucho mayor que con valores más altos, tanto que podría considerarse crecimiento exponencial. La causa de esto se explicará en el próximo apartado.

El análisis de la segunda parte del partido tarda considerablemente más que la primera, y sus archivos resultantes también ocupan más, por lo que contendrán más datos, lo que significa que el balón ha pasado más tiempo en disparos que se dirigían a una portería. Puesto que en esta consulta se analizan los tiros a puerta, que son fruto únicamente de las decisiones que tomen los jugadores sobre tirar o no, y la dirección que lleven estos disparos, podría considerarse fruto del azar que en la segunda parte haya habido más tiros a puerta, o más duraderos, y que eso provoque que haya más datos que analizar y se tarde más en hacerlo.

Como en la segunda consulta, hay que tener en cuenta que en el enunciado del proyecto se menciona que al final de la primera parte, el sensor del balón dejó de funcionar, y no se pudieron medir sus datos durante los últimos dos minutos y medio. Esta puede ser otra

causa de que los análisis de la segunda mitad del partido tarden más que los de la primera, ya que ésta tiene menos datos que analizar.

6.3.5 Precisión utilizada

En términos generales, cuanto mejor es la precisión, peor es el rendimiento, algo que era de esperar. Pero esta relación no es directamente proporcional, porque el valor de precisión es un límite de corte, que hace que se tengan o no en cuenta determinados momentos del juego, y estos no siguen ningún patrón, son fruto del azar del propio juego.

En los mayores valores de precisión (peor precisión), el rendimiento no se ve afectado en gran manera, porque sólo se detectan los golpeos más fuertes, y la cantidad de datos a analizar es menor.

A medida que se va reduciendo el valor de la precisión (mayor precisión), el número de datos a analizar aumenta cada vez en mayor medida, algo lógico teniendo en cuenta que el límite de corte es menor y los valores que lo pasan son todos los que ya lo habían pasado más los nuevos.

La pelota se encuentra la mayoría del tiempo a una aceleración baja, por lo que los valores más bajos para la precisión (próximos a los 55 m/s^2 recomendados en el enunciado del proyecto) van a hacer que se analicen una cantidad de datos mucho mayor a valores más altos, lo que a su vez va a repercutir de forma igualmente negativa en el rendimiento.

Establecer el valor de corte de la precisión en un número bajo también tiene otra parte negativa, además del rendimiento, que son los toques inexistentes y los duplicados.

La aceleración de la pelota oscila mucho, debido a que la tasa de medición de los datos es muy alta (2000 veces por segundo) y la pelota está casi siempre en movimiento. Esto hace que los pequeños botes al caer o al rodar por el campo puedan alcanzar el valor de corte de la precisión si es demasiado bajo y, por lo tanto, ser interpretados como golpeos al balón si hay un jugador cerca.

Con una tasa de medición tan alta, cuando se golpea el balón, su aceleración oscila durante unos instantes. Si una parte de los valores entre los que oscila están por debajo del valor de la precisión y la otra parte de los valores está por encima, cada vez que oscile la aceleración en ese corto periodo de tiempo será interpretado como un nuevo toque al balón, lo que, sin aplicar ningún filtro, haría que la cantidad de golpeos al balón recogidos por jugador fuera escandalosamente grande e irreal.

Por esto, aunque principalmente ocurre con los valores de precisión más pequeños, en la consulta 2 se ha añadido un filtro para todos los valores que hace que, una vez detectado un golpeo, se ignoren todos los golpeos siguientes (si los hay) durante los 0.1 segundos siguientes.

Se ha escogido la cifra de 0.1 segundos porque es un valor suficientemente grande como para eliminar los posibles falsos positivos y suficientemente pequeño como para no obviar

casos donde pueda haber varios golpeos muy seguidos, como por ejemplo en una posible pelea por el balón entre dos o más jugadores.

Una de las metas perseguidas ha sido encontrar el valor para la precisión que ofrezca el mejor equilibrio entre rendimiento y calidad de los resultados. Para ello se han establecido distintos valores de precisión y se han probado en distintas condiciones.

La precisión entra en juego en las consultas 2 y 4, que son las que tienen en cuenta los golpeos al balón; una para calcular la posesión y la otra para detectar los tiros a puerta.

Tanto en la consulta 2 como en la 4 la precisión usada es el factor más determinante para el rendimiento. Viendo las gráficas de ambas consultas se ve que, a partir del valor de precisión 100 para abajo, el tiempo utilizado crece en mayor medida que la precisión, al contrario que en valores previos más altos de 100. Podría considerarse el valor de precisión 100 como punto de inflexión para el rendimiento.

Analizando los resultados obtenidos con los distintos valores de precisión, también se ve que es a partir de esa precisión los resultados tienen ya una buena calidad y son aceptables.

Para el caso en el que se han realizado las pruebas, que ha sido en una máquina virtual en un solo ordenador de sobremesa usando sólo 4 núcleos del procesador, el valor de precisión más adecuado es 100 por tener el mejor rendimiento, es decir, el mejor equilibrio entre la precisión de los resultados y el tiempo empleado en obtenerlos.

No obstante, en la mayoría de casos en los que se requiera o se vaya a usar este programa, seguramente se cuente con más ordenadores y de más potencia, por lo que, dada la escalabilidad del programa respecto a la cantidad de procesadores disponibles, lo más recomendable sería disminuir el valor de precisión y así aprovechar la potencia y paralización extras para obtener resultados más precisos y tardando menos tiempo en hacerlo.

6.4 Rendimiento reducido a causa de vulnerabilidades de Intel

En enero de 2018, se encontraron dos vulnerabilidades, llamadas *Meltdown* y *Spectre*, en la arquitectura de los procesadores Intel de los últimos años. Esto hizo que a los pocos días Microsoft lanzara una actualización para Windows 10 para parchear estas vulnerabilidades, modificando el modo en que el sistema operativo utiliza el procesador. Las consecuencias han repercutido negativamente en el rendimiento de este proyecto en las fases de implementación y pruebas, por lo que se ha considerado oportuno explicar en qué consiste este problema y cómo ha afectado al proyecto.

La vulnerabilidad *Meltdown* quebranta el aislamiento más fundamental entre las aplicaciones de usuario y el sistema operativo. Este ataque permite que un programa acceda a la memoria y, por lo tanto, también a los secretos, de otros programas y del sistema operativo.

La vulnerabilidad *Spectre* rompe el aislamiento entre aplicaciones distintas. Permite a un atacante engañar a programas que funcionan correctamente y usan buenas prácticas para filtrar sus secretos. De hecho, las comprobaciones de seguridad de estas buenas prácticas en realidad incrementan el área de ataque y pueden hacer a las aplicaciones más susceptibles a *Spectre* [24].

El ordenador utilizado para la ejecución de este proyecto cuenta con una CPU Intel y tiene Windows 10 como sistema operativo base, por lo que le afectan ambas vulnerabilidades, aunque la única que afecta directamente al proyecto es *Meltdown*, por razones que se explican a continuación.

Meltdown afecta a la virtualización del procesador, haciendo que las aplicaciones que hagan uso de la capacidad de virtualización del procesador, puedan acceder a algunos espacios de memoria asignados a otras aplicaciones que también usen la virtualización.

Este proyecto se ha realizado enteramente utilizando una máquina virtual para su ejecución, por lo que cualquier modificación en las características de virtualización disponibles para el procesador va a afectar a la ejecución del programa.

Si bien la vulnerabilidad *Meltdown* afecta al procesador que se ha utilizado, la vulnerabilidad no es relevante en sí, puesto que el ordenador utilizado es de uso particular y no se ha utilizado más de una máquina virtual, por lo que no hay peligro de acceso indebido o robo de información.

No obstante, el parche oficial lanzado por Windows para mitigar la vulnerabilidad sí que afecta directamente a la ejecución del programa desarrollado en este proyecto.

La actualización que lanzó Windows corrige las vulnerabilidades desactivando las características de la CPU de las que se aprovechaban, un método eficaz pero que tiene consecuencias negativas respecto al rendimiento del procesador, que al no poder utilizar el 100% de su capacidad, disminuye su rendimiento.

Habitualmente, el *kernel* (núcleo) de Windows está en todos los espacios de memoria para no tener que cambiar de modo usuario al modo *kernel* constantemente con cada operación. El parche de Windows hace que el *kernel* esté ahora en un espacio de memoria aislado, y se tenga que pasar a modo *kernel* para cada operación, lo cual añade un retardo importante y hace que el rendimiento de la CPU se reduzca [25].

El ordenador en el que se ha ejecutado el código de este proyecto instala las actualizaciones de Windows entre quince y treinta días después de que se lancen las actualizaciones oficialmente para evitar actualizaciones tempranas que puedan dar problemas, por lo que, a la hora de hacer las pruebas de rendimiento, la actualización

referente a las vulnerabilidades ya estaba instalada, y no se ha podido medir si realmente ha habido una merma en el rendimiento con dicha actualización. Además, dado que el código se ha ido optimizando con el paso del tiempo, el rendimiento no ha hecho más que ir aumentando conforme pasaban los días hasta hacer las pruebas finales.

7. CONCLUSIONES Y FUTURAS LÍNEAS DE TRABAJO

7.1 Conclusiones

El objetivo de este proyecto era diseñar e implementar un sistema *big data* utilizando Spark que permitiese procesar y analizar un conjunto de datos provenientes de un partido de fútbol y realizar sobre ellos varias consultas para obtener como resultado diversas estadísticas que puedan ser utilizadas para observar con más detalle el rendimiento de los jugadores y los equipos durante el partido.

Se ha buscado que el sistema implementado sea eficiente y tenga un buen rendimiento, además de ser escalable en caso de utilizarse en equipos de mayor potencia. Tras implementarlo y realizar las pruebas de rendimiento se puede concluir que se han conseguido cumplir los objetivos especificados.

El sistema implementado es capaz de leer un gran volumen de datos, procesarlo y aplicar sobre los datos las consultas especificadas por el usuario con los parámetros que elija, para obtener los resultados detallados y que sean de utilidad.

7.2 Futuras líneas de trabajo

Algunos posibles añadidos con los que se podría continuar este proyecto son:

- Tener en cuenta la física del balón: Se podrían incluir algunas leyes físicas en los cálculos para, teniendo los datos del balón, prever la trayectoria de los disparos a puerta y los disparos con efecto recién se produzca el golpeo.
- Computación distribuida: Dada la buena escalabilidad de Spark, el programa realizado en este proyecto podría utilizarse en varios ordenadores a la vez o en un clúster, usando computación distribuida.
- Funcionamiento en tiempo real: Este programa también podría intentar utilizarse en tiempo real, durante un partido, analizando los datos en cuanto van llegando de los sensores y calculando y mostrando los resultados al instante.
- Inteligencia artificial: Con los datos proporcionados y las estadísticas que se pueden obtener, podría utilizarse este programa con una inteligencia artificial, que usara algoritmos de aprendizaje y predicción para saber cosas como por donde va a moverse cada jugador, qué jugadores van a ir a por un balón o por qué zona es más probable que ataque un equipo.

8. PLANIFICACIÓN

8.1 Fases de desarrollo

- Estudio del problema y planteamiento de las necesidades: En esta fase se contempla el proyecto pensando en su totalidad. Se analizan las necesidades y se piensan los pasos necesarios para cubrirlas.
  Tiempo: 15 días.
- Estudio de las tecnologías que se utilizarán: Se estudia cómo funcionan las herramientas y tecnologías que se usarán en el proyecto, principalmente Spark. Se estudia su documentación y se buscan sugerencias de uso y ejemplos.
  Tiempo: 20 días.
- Diseño del sistema: Se plantean las distintas propuestas de desarrollo utilizando las tecnologías escogidas y la arquitectura que tendrá el sistema para determinar el diseño a seguir.
  Tiempo: 15 días.
- Configuración del entorno de implementación: Instalación y configuración del sistema y las herramientas que se van a usar (VirtualBox, Ubuntu y Spark).
  Tiempo: 7 días.
- Implementación: Desarrollo del código para que funcione correctamente el programa diseñado.
  Tiempo: 90 días.
- Configuración del entorno de prueba: Se reorganiza el código para que sea más fácil cambiar sus parámetros de configuración y se determinan las distintas configuraciones que se usaran en las pruebas.
  Tiempo: 7 días.
- Ejecución de pruebas de rendimiento: Se realizan diferentes pruebas con distintas configuraciones para medir el rendimiento de cada configuración.
  Tiempo: 60 días.
- Análisis de los resultados: Se comparan los resultados de las distintas pruebas hechas para sacar conclusiones y realizar gráficas comparativas.
  Tiempo: 5 días.
- Preparación de la memoria: Documentación de todo el trabajo realizado en el proyecto.
  Tiempo: 60 días.

8.2 Diagrama de fases de ejecución

En el siguiente diagrama de Gantt en la Figura 27 se ilustra la planificación seguida.

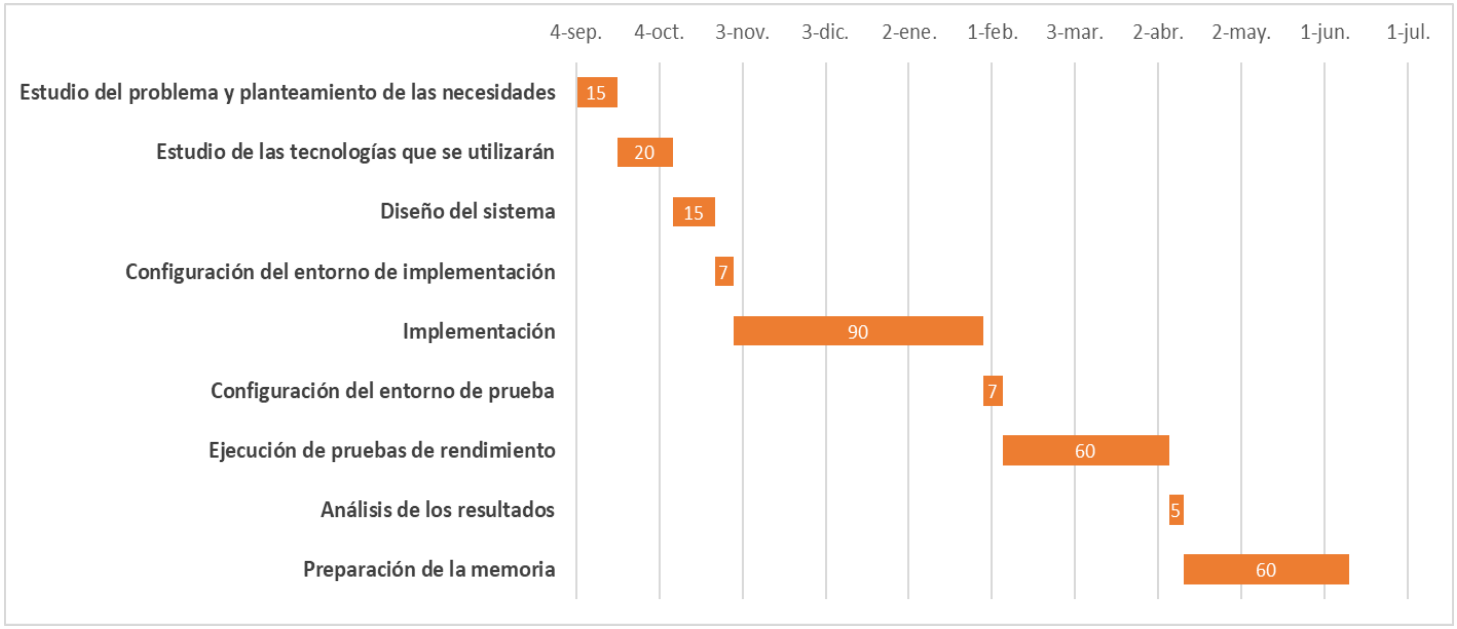

Figura 27: Diagrama de fases de ejecución

9. PRESUPUESTO E IMPACTO SOCIOECONÓMICO

9.1 Medios empleados

A continuación, se van a listar los medios empleados en este proyecto.

Recursos humanos:

- 1 ingeniero junior
- 1 ingeniero senior

Recursos materiales

- 1 PC de sobremesa de gama media-alta (CPU i7-4770, RAM 16GB, HDD 1TB)
- 1 monitor Samsung
- 1 teclado Logitech
- 1 ratón Logitech
- 1 disco duro externo (WD 3TB)

Software y licencias

- Oracle VirtualBox 5.1
- Ubuntu 17.04
- Apache Spark 2.2
- Notepad++ v6
- Microsoft Office 2016

Recursos inmateriales

- Conexión a internet de fibra óptica (9 meses)

9.2 Presupuesto del trabajo

En este proyecto han participado dos personas:

- 1 ingeniero junior, con dedicación de media jornada.

- 1 ingeniero senior, con dedicación del 10% de la jornada.

A continuación, en las Tablas 3 y 4 están reflejadas las horas dedicadas por cada uno y sus distintos salarios.

FASE	DÍAS	HORAS INGENIERO JUNIOR	HORAS INGENIERO SENIOR
Estudio del problema y planteamiento de las necesidades	15	60	12
Estudio de las tecnologías que se utilizarán	20	80	16
Diseño del sistema	15	60	12
Configuración del entorno de implementación	7	28	6
Implementación	90	360	72
Configuración del entorno de prueba	7	28	6
Ejecución de pruebas de rendimiento	60	240	48
Análisis de los resultados	5	20	4
Preparación de la memoria	60	240	48
TOTAL	279	1116	224

Tabla 3: Recursos humanos

CONCEPTO	CANTIDAD	HORAS	COSTE POR HORA	COSTE TOTAL
Ingeniero junior	1	1116	16	17856
Ingeniero senior	1	224	30	6720
			TOTAL	24576

Tabla 4: Total recursos humanos

Las tasas de depreciación son las marcadas oficialmente por la Agencia Tributaria Española.

En las Tablas 5, 6 y 7 se muestran los costes de los demás recursos utilizados y al final, en la Tabla 8, se muestra el coste total del proyecto.

CONCEPTO	CANTIDAD	PRECIO	MESES DE USO	DEPRECIACIÓN	MESES DE DEPRECIACIÓN	COSTE TOTAL
PC de sobremesa	1	650	9	25%	48	121.88
Monitor	1	250	9	25%	90	0
Teclado	1	15	9	25%	84	0
Ratón	1	52	9	25%	36	13
Disco duro externo	1	90	6	25%	6	78.75
					TOTAL	200.63

Tabla 5: Recursos materiales

CONCEPTO	CANTIDAD	PRECIO	MESES DE USO	DEPRECIACIÓN	MESES DE DEPRECIACIÓN	COSTE TOTAL
Oracle VirtualBox 5.1	1	0	9	33%	9	0
Ubuntu 17.04	1	0	9	33%	9	0
Apache Spark 2.2	1	0	9	33%	9	0
Notepad++ v6	1	0	9	33%	21	0
Microsoft Office 2016	1	0	9	33%	21	0
					TOTAL	0

Tabla 6: Software y licencias

CONCEPTO	CANTIDAD	MESES DE USO	COSTE	COSTE TOTAL
Conexión a internet	1	9	80	720
			TOTAL	720

Tabla 7: Recursos inmateriales

CONCEPTO	COSTE
Recursos humanos	24576
Recursos materiales	200.63
Software y licencias	0
Recursos inmateriales	720
TOTAL	25496.63

Tabla 8: Presupuesto total

9.3 Impacto socioeconómico

Este programa puede ser utilizado por personas u organizaciones para obtener una ventaja competitiva, permitiéndoles analizar el rendimiento de los equipos, mostrando las debilidades de sus propios jugadores para corregirlas, o las de los jugadores rivales para aprovecharlas.

Una futura versión de este programa que lograse poder contar con potencia suficiente para operar en tiempo real podría dar lugar también a una nueva experiencia mejorada para los espectadores, que les mostrase análisis y estadísticas interactivos durante los partidos.

Se espera que los resultados conseguidos en este proyecto propicien la adaptación progresiva de sistemas basados en eventos en tiempo real y que los datos y las consultas utilizados sirvan para evaluar sistemas basados en eventos en más ámbitos que los que abarca este proyecto.

10. LEGISLACIÓN Y MARCO REGULADOR

El *big data* se basa en el análisis de grandes volúmenes de datos. Si los datos analizados contienen información personal puede afectar a la privacidad de las personas a las que pertenecen.

Recientemente, se ha hecho obligatorio el uso de una nueva normativa que se usará tanto en España como en la Unión Europea. Se trata del nuevo Reglamento General de Protección de Datos (*General Data Protection Regulation*, en inglés). Esta normativa afecta a todas las empresas, europeas o no, que hagan uso de datos de ciudadanos europeos, además de a los propios ciudadanos que vivimos en la Unión Europea.

Este reglamento, entre otras cosas, obliga a las empresas a informarte sobre cómo van a tratar tus datos personales antes de que des tu consentimiento para ello y a informar en un plazo máximo de 72 horas si descubren que su seguridad se ha visto comprometida, tanto a las autoridades pertinentes como a los usuarios afectados. También exige renovar el consentimiento en todos los servicios que estén afectados por la nueva normativa, aunque ya se haya dado el consentimiento antes del 25 de mayo [26].

Los derechos para los ciudadanos que incluye esta nueva normativa son los siguientes [27]:

- Derecho de acceso: El ciudadano debe poder obtener información sobre sus datos personales y para qué se van a utilizar.
- Derecho de rectificación: Se debe permitir corregir o modificar los datos que sean incorrectos o incompletos.
- Derecho de supresión: Las personas deben poder exigir la eliminación de sus datos personales. También se conoce como “Derecho al olvido”.
- Derecho a la limitación del tratamiento: El ciudadano puede suspender el tratamiento de sus datos personales bajo ciertas circunstancias.
- Derecho a la portabilidad de los datos: Se debe poder obtener los datos personales recogidos de forma estructurada.
- Derecho de oposición: Los ciudadanos deben poder oponerse al tratamiento de sus datos personales.
- Derecho a no ser objeto de decisiones individualizadas: Las personas deben poder oponerse a las decisiones tomadas tratando sus datos de manera automatizada.

10.1 Estándares técnicos

El *big data* aún lleva poco tiempo utilizándose en empresas, por lo que de momento sólo existe un estándar, que se desarrolló en 2015. Éste consiste en la Recomendación UIT-T Y.3600 de noviembre de 2015, llamada “Big data – Cloud computing based requirements

and capabilities", aprobada por los miembros de la Unión Internacional de Telecomunicaciones, un organismo dependiente de la ONU [4].

Este documento proporciona requisitos, prestaciones y casos de uso del *big data* en la computación en la nube y su contexto de sistema.

11. EXTENDED ABSTRACT

11.1 Introduction

Collecting data for later analysis and information obtainment is something that is increasingly done in all areas. Collecting data is not difficult, since most people have different accounts on the internet for different services. The problem that arises is how to analyze such a large amount of data. You have to think about how to do it so that it takes as little time as possible and you get the desired information.

For a long time, there have been tools designed for this kind of task, but in the last few years, the amount of data has been growing exponentially, so it is necessary to improve the way of getting things done, new tools are needed.

In this project, tools designed for it will be used, such as Apache Spark, and will be applied to sports events, more precisely, to the monitoring of a football game, a scenario from which you can obtain lots of data and statistics that show the performance of the teams and their players in different aspects of the game.

More and more modern technologies are being applied to the world of sports. An example at present is the implementation of the VAR (Video Assistant Referee), a system designed to help referees during matches that allows them to review a specific move on a monitor a few seconds after it happened [1].

Systems like this start from the premise that people are not perfect and we cannot be aware of everything that happens, and that is why sometimes we make mistakes or proceed in a wrong way. This, in addition to being applied to the referees, can also be applied to the coaches, which the only information they have is what they and their assistants see. If they could see their team's statistics, they would surely fine-tune their tactics. The spectators who follow the games on television are already enjoying features of this kind, when at certain moments during the game, especially at the end of each part, diverse data of the teams and players performance are displayed on the screen [2].

Getting that information is not an easy task. For this, players must be monitored during the match, which generates a large data flow that is expensive to analyze in a short time. This is what this project will be about.

11.2 Objectives

The objective of this project is to create a *big data* system efficient in analyzing a large amount of data derived from a football match to obtain statistics regarding the course of the game and the players performance, and thus demonstrate the effectiveness of applying

event-based systems to provide a set of analytics for the teams coaches and managers and for the spectators of the event as well.

In this final degree project, we will look for a solution to the DEBS 2013 Grand Challenge [3], applying the basics of *big data* to the provided data to try to reach the best possible solution.

The defined objectives are:

- Study of the current situation and context and the *big data* tools available.
- Analysis of the problem to solve and the provided data.
- Design of a *big data* system adapted to the project requirements.
- Implementation of the designed *big data* system.
- Planning and execution of performance tests with different configurations.
- Performance analysis based on the tests results.
- Obtaining the conclusions derived from the results analysis and planning of future lines of work for a possible continuation of the project.

11.3 Document structure

- Chapter 1: Introduction
  This section exposes the motivation of the realized project and its context.
- Chapter 2: Objectives
  This section lists the proposed objectives.
- Chapter 3: Document structure
  This section describes the structure of this abstract.
- Chapter 4: State of the art
  This section contains information about the technologies and tools used in this project.
- Chapter 5: System design
  This section shows the design of the system, its components and the interactions between them.
- Chapter 6: Performance tests
  This section contains the performance tests carried out, their results and an analysis of them.
- Chapter 7: Conclusions and future lines of work
  This section exposes the conclusions obtained at the end of the project and possible ways to continue the work done.

11.4 State of the art

11.4.1 Big Data

In the current times, greater volumes of data are being generated and stored. The *big data* consists of processing all this data to obtain additional information to what they already offer. Thanks to this processing of large volumes of data, statistics are obtained that may be relevant especially for large companies, which may know better the use of their services or their users behavior.

There are four main concepts that define *big data*, known as the 4V's:

- Volume: *Big data*, as its name suggests, is synonymous with large amounts of data. It is its main characteristic.
- Velocity: With technological advances, large amounts of data can be analyzed and processed faster, even in real time, which is one of the reasons why the use of *big data* is expanding.
- Variety: The data used for *big data* can come from very different sources, be differently structured or in different formats.
- Veracity: We must ensure that the data used is true. Otherwise, the results would not be reliable.

In today's world, we are always surrounded by technology, and we interact more with it each day, for different things. Every interaction or operation that we do is monitored and stored so that it can be used as information to know us better. This is known as digital footprint.

There are different types of data analysis, which are classified into four different categories, according to the purpose for which they are made [9]:

- Prescriptive analysis: To detect patterns and improve performance in the face of problems in the future.
- Predictive analysis: Consists in predicting what will happen
- Diagnostic analysis: Try to determine why something has happened
- Descriptive analysis: It is to make data mining, extract useful and relevant information from large amounts of data.

Data mining is the analytical part of the KDD process (Knowledge Discovery in Databases), and has five phases: Selection, pre-processing, transformation, data mining and evaluation. A process that consists of finding patterns in large datasets using statistics, machine learning and databases. The main objective of the process is to extract information from a data set to transform it and structure it so that it can be useful later on [10].

11.4.2 Apache Spark

Apache Spark is an open code cluster-computing framework, which architecture is based on Resilient Distributed Datasets (RDD's), and makes easy to use iterative and data analysis algorithms.

Spark makes use of a cluster manager, so it supports several types, and needs also a distributed storage system, but is compatible with a great variety too.

Spark has a local (pseudo-distributed) mode that does not use distributed storage and can be used in an environment that consists of a single system, using the local file system and running on a single CPU, with one executor per core [19] [20].

The two most important components of Spark are:

- Spark Core: the main component, which provides distributed task dispatching, scheduling, and basic input/output functionalities.
- Spark SQL: a component of Spark that works on its Core and allows handling DataFrames, structured or semi-structured datasets, using languages such as Scala, Java, Python and SQL [22].

11.5 System design

11.5.1 Architecture

This system has been designed to run on a single computer. The system works in the virtual machine on which it has been installed, on the hardware of the host desktop computer, which consists of a 7200 RPM hard drive, 16 GB of RAM and an Intel Core i7-4770 CPU with 4 physical cores and 8 virtual cores. Spark will use 1 executor per CPU core, so it will distribute the work load between 4 executors, because the virtual machine only allows to use half of the computer resources.

11.5.2 Operating cores

The designed system has a single user role that is administrator. This will be responsible for making use of all the system's functionalities and executing the queries. These functionalities are:

- Data storage: The system needs the files with the statistics collected by the sensors in order to process them and obtain the necessary data for the queries. This data come in a plain text file and will be saved on the virtual machine hard drive to be

able to access them. The game interruption data collected by the referee will also be saved in CSV format.

- Data processing and queries execution: The stored files need to be read and processed to select the data that is really needed for queries. Once the query and the data to be used are selected, the processing is executed followed by the query. Once the query is finished, the results are written into a text file in the folder designated for it on the virtual machine hard drive, so the user can access them.

11.5.3 System modules

The system consists of three different parts or modules that have to interact with each other for the complete functioning of the system:

- User: Is the system administrator and the one in charge of interacting with him. The user has to save the data on the hard disk so that Spark can access it. The user must also start Spark and specify the parameters that will be used and the query that will be executed.
- Storage: Where the data and the results obtained are stored. In the system used, it refers to the virtual hard disk used by the virtual machine. This component is responsible for storing all the files that will be used for queries and the files with the results.
- Spark: It processes the data, executes queries and saves their results. This component is responsible for reading all data, processing it to obtain the specific required data according to the specified query and its entered parameters, executing said query following the parameters and, once finished, saving the results in a file in the storage.

11.6 Performance tests

11.6.1 Test environments

To check the performance of the program with different input parameters that give different results in the queries, there have been used different combinations of:

- Different players.
- Different halves of the game.
- Different periods of time.
- Different precision values to detect ball hits.

11.6.2 General results

In general, the results regarding the different combinations of parameters are as expected. The parameters that increase the period of time to analyze and those that improve the accuracy have a negative impact on performance, something logical, since they increase the size of the data to be analyzed and the query takes longer to complete.

Next, each of the four queries are presented with all the parameters used and their results, and how they are affected by the precision values.

11.6.3 Query 1: Running analysis

The different parameters used are:

- Halves: 1, 2.
- Periods of time (minutes): [1,15], [1,30].
- Players: A1, A3, B1, B3.
- Players SID: 16 (A1), 19 (A3), 63 (B1), 67 (B3).

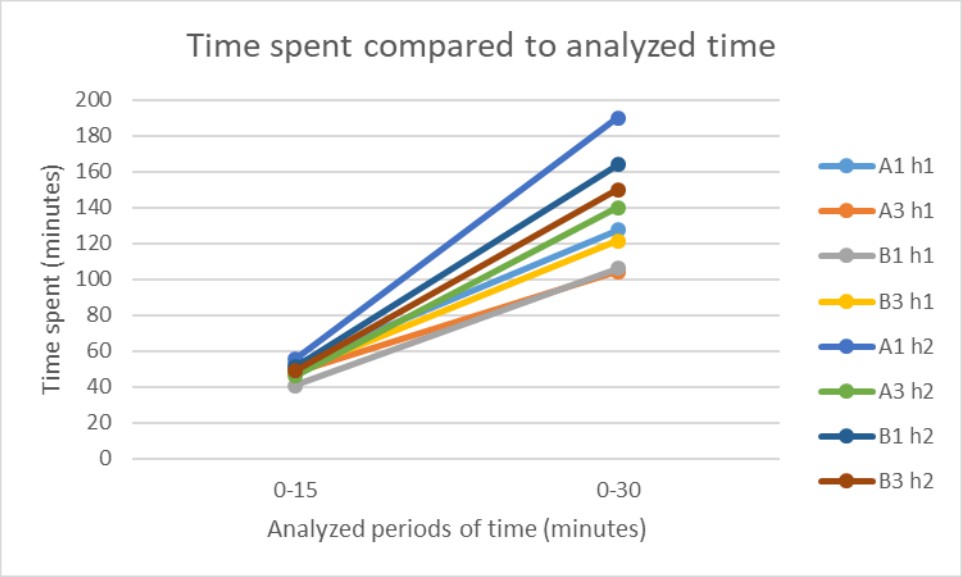

Figure 1: Query 1 results graph

The results of the performance tests show that it takes longer to analyze each complete half of the game than half of each, as it was logical to expect. Analyzing each complete half always takes more than twice as long, and even in the worst case, it takes more than three times. It is also seen that processing the second half of the game always takes longer than the first.

These two observations may be because, in order to process the entire original file, which is what is needed for the second half, it is necessary to go over more than twice as much data as for the first, and in addition, the data that is necessary is further the file because it is later. This happens in a similar way when analyzing more data in the same half, even if it is the first.

All this adds extra computation time regarding the analysis of the data that is first in the file, especially when analyzing large quantities.

11.6.4 Query 2: Ball possession

The different parameters used are:

- Halves: 1, 2.
- Periods of time (minutes): [1,10], [1,20], [1,30].
- Precision values (m/s^2): 300, 250, 200, 150, 100, 75, 55.

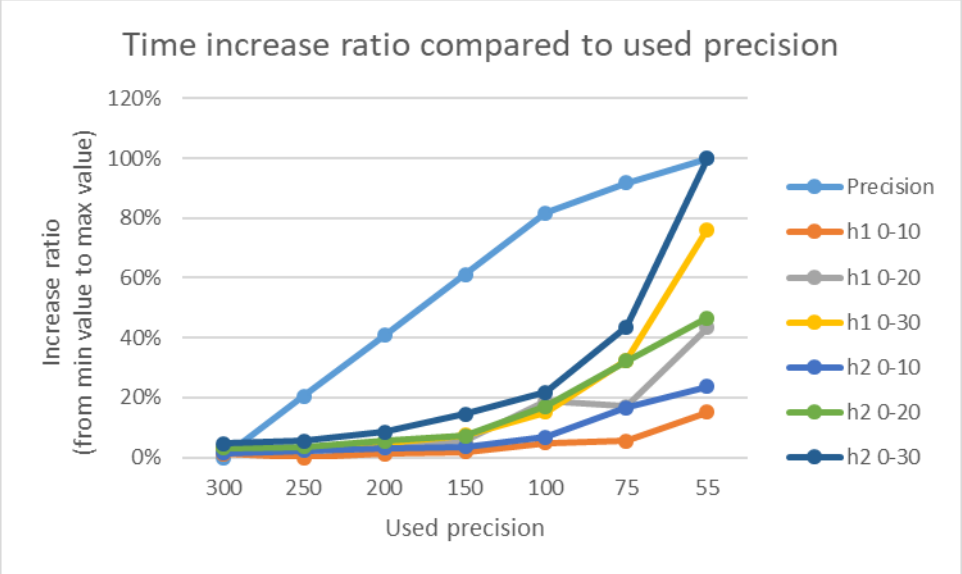

Figure 2: Query 2 results graph

In the results graph it is clear how the performance of the query is scaled to the amount of data analyzed and the precision used, and depends also on the half of the analyzed game.

The more minutes to analyze, the more data to process, and therefore, it takes more time. It is also seen that what most influences the time it takes is precision. With high precision values (low precision), times are hardly affected, but as that value is reduced, time increases greatly, even seems to increase exponentially in cases with more data to analyze.

It is also seen that the analysis of the first half always take less time than the equivalent analysis of the second half. Most likely, this is due to the same causes as in the first query, where the same thing happened. To analyze the second half you have to go over more than twice the data for the first half and these data are farther in the file. However, it could be, perhaps, because the first half of the match has less data to analyze because of a ball sensor fault in the last minutes.

11.6.5 Query 3: Heat map

The different parameters used are:

- Halves: 1, 2.
- Players: A1, A2, A3, B1, B2, B3.
- Periods of time for the A team (minutes): [1,15], [1,30].
- Periods of time for the B team (minutes): [1,10], [1,20], [1,30].

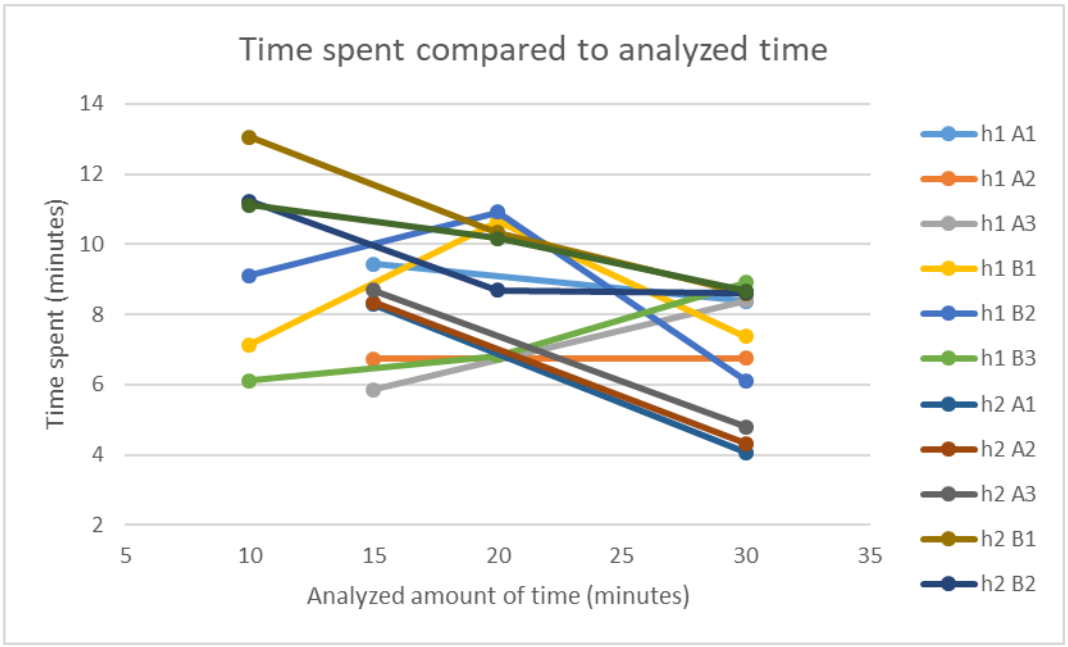

Figure 3: Query 3 results graph

Looking at the graph, the results are mixed. The only appreciable trend is that time is reduced by analyzing more data. The only possible explanation is that it takes less time because Spark has previous data in cache and thus the program does not need to search for it again.

11.6.6 Query 4: Shots on goal

The different parameters used are:

- Halves: 1, 2.
- Precision values (m/s^2): 300, 250, 200, 150, 100, 75, 55.

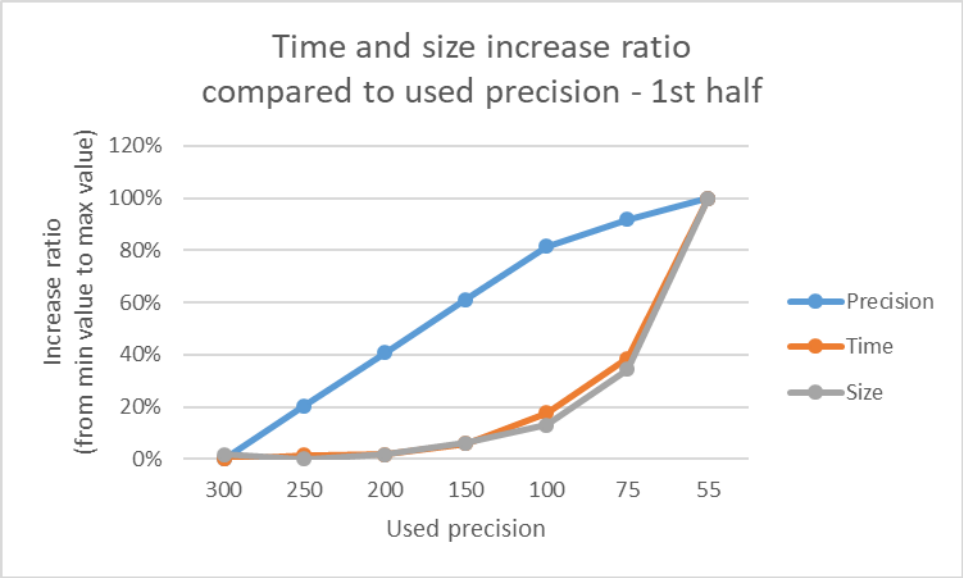

Figure 4: Query 4 results graph 1

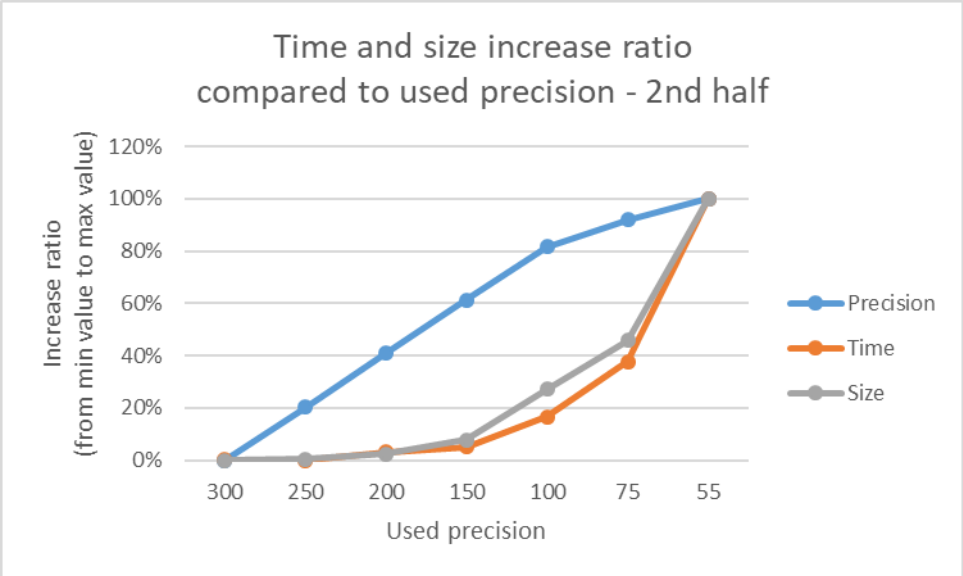

Figure 5: Query 4 results graph 2

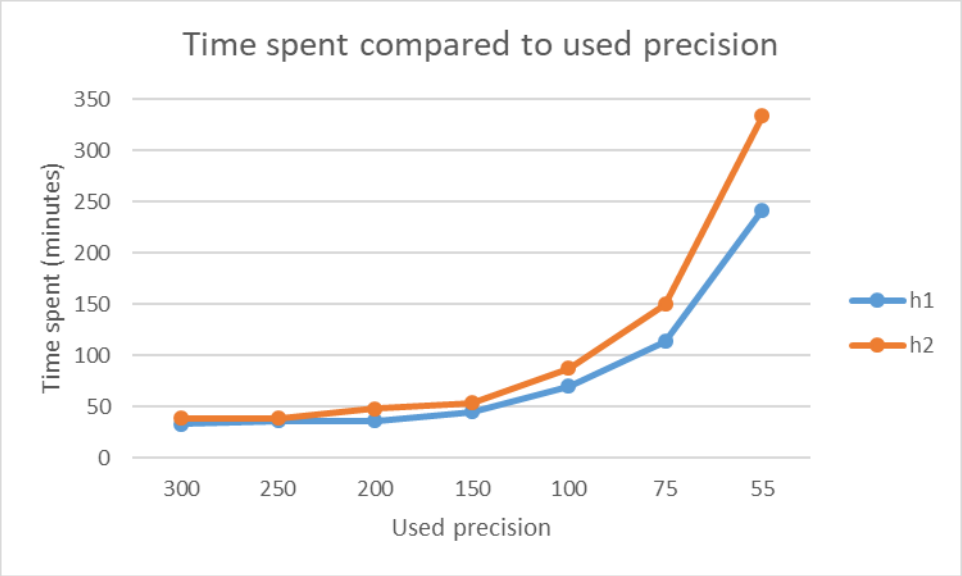

Figure 6: Query 4 results graph 3

In the graphs obtained from the results of the performance tests, can be easily seen that the size of the generated file is almost identically proportional to the time it takes to process the query.

This makes sense, since the time it takes to process the query depends mainly on the amount of processed data, which is what is shown in the resulting file, so it is logical to think that the time will increase almost equivalent to the increase of the data and the size of the file.

It is also observed how, both the time used and the size of the file, increase with the precision used. With the lower precision values (higher precision) they increase in a much higher proportion than with higher values, so much that it could be considered exponential growth.

The analysis of the second half of the match takes considerably longer than the first, and its resulting files weight more too, so they will contain more data, which means that the ball has spent more time in shots that went to a goal. Since this query analyzes the shots on goal, that are just the result of the decisions that take the players on kicking or not, and the direction that take these kicks, could be considered a random result that in the second half there were more shots on goal, or more durable, and that causes there is more data to analyze and it takes longer to do it. However, there must be taken into account that, as in the second query, it could be because the first half of the match has less data to analyze because of a ball sensor fault in the last minutes.

11.6.7 Used precision

In general, the better the accuracy, the worse the performance, which was expected. But this relationship is not directly proportional, because the precision value is a cutoff limit,

which means that certain moments of the game are taken into account or not, and these do not follow any pattern, they are a random result of the game itself.

With the highest precision values (worse accuracy), the performance is not greatly affected, because only the strongest hits are detected, and the amount of data to analyze is less.

As the precision value is reduced (better precision), the data size to analyze increases more and more, something logical considering that the cutoff limit is lower and the values that pass it are all that had previously passed plus the new ones.

The ball is most of the time at a low acceleration, so the lower values for accuracy (near the recommended 55 m/s^2 in the project statement) will make it analyze a much greater amount of data at higher values, which, in turn, will have an equally negative impact on the performance.

Setting the cutoff value of the precision in a low number also has another negative fact, in addition to the performance, which are nonexistent touches and duplicates.

The acceleration of the ball oscillates very much, because the measurement rate of the data is very high (2000 times per second) and the ball is always in motion. This means that small bounces when falling or rolling down the field can reach the cutoff value of the accuracy if it is too low and, therefore, be interpreted as someone hitting the ball if there is a player nearby.

With such a high measurement rate, when the ball is hit, its acceleration oscillates for a few instants. If one part of the values between which oscillates are below the value of the precision and the other part of the values is above, each time the acceleration oscillates in that short period of time it will be interpreted as a new touch to the ball, which, without applying any filter, would make the number of hits collected enormously large and unreal.

For this reason, although this mainly affects only the smallest precision values, in the second query a filter has been added for all the values, which means that, once a hit is detected, all subsequent hits (if any) during the next 0.1 seconds are ignored.

The figure of 0.1 seconds has been chosen because it is a large enough value to eliminate possible false positives and small enough to avoid cases where there may be several hits in a row, such as in a possible fight for the ball between two or more players.

One of the goals pursued has been to find the precision value that offers the best balance between performance and quality of results. For this purpose, different precision values have been established and tested under different conditions.

The precision comes into play in queries 2 and 4, which are those that take into account the ball hits; one to calculate possession and the other to detect shots on goal.

In both query 2 and 4, the precision used is the most determining factor for performance. Viewing the graphs of both queries shows that, from the precision value 100 and below, the time taken grows greater than the precision, unlike in previous values higher than 100.

In the scenario in which the tests have been carried out, which has been a virtual machine in a single desktop computer using only 4 processor cores, the most suitable precision value is 100 for having the best performance, that is, the better balance between the accuracy of the results and the time taken to obtain them.

11.7 Conclusions and future lines of work

11.7.1 Conclusions

The objective of this project was to design and implement a *big data* system using Spark that would allow to process and analyze a set of data coming from a football game and make several queries about them to obtain as a result various statistics that can be used to observe the players and teams performance during the game in more detail.

It has been sought that the implemented system is efficient and has a good performance, as well as being scalable if used in higher power systems. After implementing it and performing the performance tests, it can be concluded that the specified objectives have been achieved.

The implemented program is capable of reading a large volume of data, processing it and applying the queries specified by the user to the data, with the chosen parameters, to obtain detailed and useful results.

11.7.2 Future lines of work

Some possible additions with which this project could be continued are:

• Take into account the ball physics: Some physical laws could be included in the calculations to predict the trajectory of the shots on goal and the bending kicks when the hitting occurs.

• Distributed computing: Given the good scalability of Spark, the program developed in this project could be used on several computers at the same time or a cluster, using distributed computing.

• Real-time operation: This program could also be used in real time, during a game, analyzing the data as soon as they arrive from the sensors, and calculating and showing the results instantly.

• Artificial intelligence: With the data provided and the statistics that can be obtained, this program could be used with artificial intelligence, which will use learning and prediction algorithms to know things like where each player is going to move, which players will go towards a ball or which area a team is more likely to attack by.

REFERENCIAS

[1] «El Mundial de Fútbol de Rusia 2018 se jugará con videoarbitraje,» CNN en Español, 3 3 2018. [En línea]. Available: http://cnnespanol.cnn.com/2018/03/03/el-mundial-de-futbol-de-rusia-2018-se-jugara-con-videoarbitraje/.

[2] @beINSPORTS, "Check out the match stats from tonight's @ChampionsLeague action. #UCL #AtletiFCB," Twitter, 27 4 2016. [Online]. Available: https://twitter.com/beinsports/status/725424541544108038.

[3] debs, "DEBS 2013 Grand Challenge: Soccer monitoring," DEBS, 22 8 2016. [Online]. Available: http://debs.org/debs-2013-grand-challenge-soccer-monitoring/.

[4] ITU, "ITU-T Recommendations," International Telecommunications Union, 6 11 2015. [Online]. Available: https://www.itu.int/itu-t/recommendations/rec.aspx?rec=12584.

[5] J. C. L. López, «La moda del Big Data: ¿En qué consiste en realidad?,» El Economista, 27 2 2014. [En línea]. Available: http://www.eleconomista.es/tecnologia/noticias/5578707/02/14/La-moda-del-Big-Data-En-que-consiste-en-realidad.html.

[6] M. Walker, "Big Data Value," Information Catalyst, 2013. [Online]. Available: http://informationcatalyst.com/vision-experience/big-data-value/.

[7] A. Nordrum, "Popular Internet of Things Forecast of 50 Billion Devices by 2020 Is Outdated," IEEE Spectrum, 18 8 2016. [Online]. Available: https://spectrum.ieee.org/tech-talk/telecom/internet/popular-internet-of-things-forecast-of-50-billion-devices-by-2020-is-outdated.

[8] "Digital footprint," Wikipedia, the free encyclopedia, 14 5 2018. [Online]. Available: https://en.wikipedia.org/wiki/Digital_footprint.

[9] A. Bekker, "4 types of data analytics to improve decision-making," ScienceSoft, 11 7 2017. [Online]. Available: https://www.scnsoft.com/blog/4-types-of-data-analytics.

[10] "Data mining," Wikipedia, the free encyclopedia, 20 5 2018. [Online]. Available: https://en.wikipedia.org/wiki/Data_mining.

[11] "Cross-industry standard process for data mining," Wikipedia, the free encyclopedia, 30 4 2018. [Online]. Available: https://en.wikipedia.org/wiki/Cross-industry_standard_process_for_data_mining.

[12] "Data, data everywhere," The Economist, 25 2 2010. [Online]. Available: https://www.economist.com/node/15557443.

[13] L. Tay, "Inside eBay’s 90PB data warehouse," iTnews, 10 5 2013. [Online]. Available: https://www.itnews.com.au/news/inside-ebay8217s-90pb-data-warehouse-342615.

[14] G. Brumfiel, "High-energy physics: Down the petabyte highway," Nature, 19 1 2011. [Online]. Available: https://www.nature.com/news/2011/110119/full/469282a.html.

[15] D. Ryan, "Sports: Where Big Data Finally Makes Sense," Huffington Post, 13 11 2016. [Online]. Available: https://www.huffingtonpost.com/dave-ryan/sports-where-big-data-fin_b_8553884.html.

[16] N. Wilson, K. Mason, M. Tobias, M. Peacey, Q. S. Huang and M. Baker, "Interpreting "Google Flu Trends" data for pandemic H1N1 influenza: The New Zealand experience," Eurosurveillance, 5 11 2009. [Online]. Available: https://www.eurosurveillance.org/content/10.2807/ese.14.44.19386-en.

[17] P. De Llano, «Una consultora que trabajó para Trump manipuló datos de 50 millones de usuarios de Facebook,» El País, 18 3 2018. [En línea]. Available: https://elpais.com/internacional/2018/03/17/estados_unidos/1521308795_755101.html.

[18] C. Kang, «Facebook admite que Cambridge Analytica accedió a los datos de 87 millones de usuarios,» The New York Times, 4 4 2018. [En línea]. Available: https://www.nytimes.com/es/2018/04/04/facebook-cambridge-analytica-87-millones/.

[19] "Apache Spark," Wikipedia, the free encyclopedia, 19 5 2018. [Online]. Available: https://en.wikipedia.org/wiki/Apache_Spark.

[20] "Spark Overview - Spark 2.2.0 Documentation," Apache Spark, [Online]. Available: http://spark.apache.org/docs/2.2.0/. [Accessed 27 5 2018].

[21] Databricks, "Spark For Big Data Analytics [Part 1]," Jen Underwood - Business Intelligence And Analytics, 16 10 2016. [Online]. Available: http://www.jenunderwood.com/2016/10/16/spark-big-data-analytics-part-1/.

[22] "Spark Programming Guide - Spark 2.2.0 Documentation," Apache Spark, [Online]. Available: https://spark.apache.org/docs/2.2.0/rdd-programming-guide.html. [Accessed 27 5 2018].

[23] Databricks, "How to speed up working with Data Frames in Spark," Vivek Mangipudi Wordpress, 13 2 2017. [Online]. Available: https://vivekmangipudi.wordpress.com/2017/02/13/how-to-speed-up-working-with-data-frames-in-spark/.

[24] "Meltdown and Spectre: Vulnerabilities in modern computers leak passwords and sensitive data.," 2018. [Online]. Available: https://meltdownattack.com/. [Accessed 27 5 2018].

[25] T. Myerson, "Understanding the performance impact of Spectre and Meltdown mitigations on Windows Systems," Microsoft, 9 1 2018. [Online]. Available: https://cloudblogs.microsoft.com/microsoftsecure/2018/01/09/understanding-the-performance-impact-of-spectre-and-meltdown-mitigations-on-windows-systems/.

[26] "2018 reform of EU data protection rules," European Commission, [Online]. Available: https://ec.europa.eu/commission/priorities/justice-and-fundamental-rights/data-protection/2018-reform-eu-data-protection-rules_en. [Accessed 27 5 2018].

[27] «Ejerce tus derechos,» Agencia Española de Protección de Datos, 25 5 2018. [En línea]. Available: https://www.aepd.es/reglamento/derechos/index.html.

ANEXOS

ANEXO A: TABLA DE DATOS DE PRUEBA Y RESULTADOS DE LA CONSULTA 1

Mitad	Periodo	Jugador	Tiempo
h1	0-15	A1	52,55
h1	0-15	A3	48,31
h1	0-15	B1	40,78
h1	0-15	B3	48,61
h1	0-30	A1	127,6
h1	0-30	A3	104,48
h1	0-30	B1	106,23
h1	0-30	B3	121,37
h2	0-15	A1	55,49
h2	0-15	A3	46,28
h2	0-15	B1	51,3
h2	0-15	B3	48,95
h2	0-30	A1	189,85
h2	0-30	A3	140,08
h2	0-30	B1	164
h2	0-30	B3	149,97

ANEXO B: TABLA DE DATOS DE PRUEBA Y RESULTADOS DE LA CONSULTA 2

Mitad	Periodo	Precision	Tiempo		Precisión	T
h1	0-10	300	20,7		0,00	0,01
h1	0-10	250	17,62		0,20	0,00
h1	0-10	200	20,97		0,41	0,01
h1	0-10	150	22,74		0,61	0,02
h1	0-10	100	31,15		0,82	0,05
h1	0-10	75	32,78		0,92	0,05
h1	0-10	55	59,78		1,00	0,15
h1	0-20	300	23,82			0,02
h1	0-20	250	23,78			0,02
h1	0-20	200	26,74			0,03
h1	0-20	150	32,86			0,05
h1	0-20	100	70,24			0,19
h1	0-20	75	65,33			0,17
h1	0-20	55	139,42			0,43
h1	0-30	300	26,44			0,03
h1	0-30	250	26,38			0,03
h1	0-30	200	30,91			0,05
h1	0-30	150	38,49			0,07
h1	0-30	100	59,98			0,15
h1	0-30	75	108,64			0,32
h1	0-30	55	231,44			0,76
h2	0-10	300	21,39			0,01
h2	0-10	250	24,15			0,02
h2	0-10	200	26,55			0,03
h2	0-10	150	27,85			0,04
h2	0-10	100	36,69			0,07
h2	0-10	75	64,18			0,17
h2	0-10	55	83,87			0,24
h2	0-20	300	26,54			0,03
h2	0-20	250	27,72			0,04
h2	0-20	200	32,91			0,05
h2	0-20	150	37,79			0,07
h2	0-20	100	65,34			0,17
h2	0-20	75	107,89			0,32
h2	0-20	55	148,57			0,47
h2	0-30	300	30,86			0,05
h2	0-30	250	33,18			0,06
h2	0-30	200	41,19			0,08
h2	0-30	150	57,98			0,14
h2	0-30	100	78,54			0,22
h2	0-30	75	139,85			0,44
h2	0-30	55	298,22			1,00

ANEXO C: TABLA DE DATOS DE PRUEBA Y RESULTADOS DE LA CONSULTA 3

Mitad	Jugador	Minutos	Tiempo
h1	A1	15	9,44
h1	A1	30	8,38
h1	A2	15	6,74
h1	A2	30	6,75
h1	A3	15	5,85
h1	A3	30	8,39
h2	A1	15	8,28
h2	A1	30	4,07
h2	A2	15	8,35
h2	A2	30	4,33
h2	A3	15	8,68
h2	A3	30	4,8
h1	B1	10	7,14
h1	B1	20	10,66
h1	B1	30	7,38
h1	B2	10	9,11
h1	B2	20	10,91
h1	B2	30	6,12
h1	B3	10	6,82
h1	B3	20	8,9
h1	B3	30	12,58
h2	B1	10	13,05
h2	B1	20	10,33
h2	B1	30	8,6
h2	B2	10	11,23
h2	B2	20	8,69
h2	B2	30	8,61
h2	B3	10	11,12
h2	B3	20	10,17
h2	B3	30	8,67

ANEXO D: TABLA DE DATOS DE PRUEBA Y RESULTADOS DE LA CONSULTA 4

Mitad	Precision	Tiempo	Tamaño resu
h1	300	32,95	4,2
h1	250	36,15	3,9
h1	200	36,54	4,2
h1	150	45,19	5
h1	100	69,81	6,2
h1	75	113,59	10
h1	55	241,8	21,6
h2	300	39,08	5,3
h2	250	38,75	5,4
h2	200	47,89	5,8
h2	150	53,65	6,9
h2	100	87,44	10,8
h2	75	150,19	14,6
h2	55	333,26	25,6